SYNGRESS®

4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP



How to Cheat at Securing Linux

The Perfect Reference for the Multitasked SysAdmin

- Discover Why "Measure Twice, Cut Once" Applies to Securing Linux
- Complete Coverage of Hardening the Operating System, Implementing an Intrusion Detection System, and Defending Databases
- Short on Theory, History, and Technical Data that Is Not Helpful in Performing Your Job

James Stanger Member of CompTIA's Linux+ Advisory Committee, Chair of Linux Professional Institute Advisory Council

SYNGRESS®

How to Cheat at Securing Linux

Mohan Krishnamurthy
Eric S. Seagren
Raven Alder
Aaron W. Bayles
Josh Burke
Skip Carter
Eli Faskha

Elsevier, Inc., the author(s), and any person or firm involved in the writing, editing, or production (collectively "Makers") of this book ("the Work") do not guarantee or warrant the results to be obtained from the Work.

There is no guarantee of any kind, expressed or implied, regarding the Work or its contents. The Work is sold AS IS and WITHOUT WARRANTY. You may have other legal rights, which vary from state to state.

In no event will Makers be liable to you for damages, including any loss of profits, lost savings, or other incidental or consequential damages arising out from the Work or its contents. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

You should always use reasonable care, including backup and other appropriate precautions, when working with computers, networks, data, and files.

Syngress Media®, Syngress®, "Career Advancement Through Skill Enhancement®," "Ask the Author UPDATE®," and "Hack Proofing®," are registered trademarks of Elsevier, Inc. "Syngress: The Definition of a Serious Security Library"™, "Mission Critical™," and "The Only Way to Stop a Hacker is to Think Like One™" are trademarks of Elsevier, Inc. Brands and product names mentioned in this book are trademarks or service marks of their respective companies.

PUBLISHED BY
Syngress Publishing, Inc.
Elsevier, Inc.
30 Corporate Drive
Burlington, MA 01803

How to Cheat at Securing Linux
Copyright © 2008 by Elsevier, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

Printed in the United States of America
1 2 3 4 5 6 7 8 9 0
ISBN-13: 978-1-59749-207-2

Publisher: Amorette Pedersen
Acquisitions Editor: Andrew Williams
Page Layout and Art: Patricia Lupien
Cover Designer: Michael Kavish
Indexer: Michael Ferreira

For information on rights, translations, and bulk sales, contact Matt Pedersen, Commercial Sales Director and Rights, at Syngress Publishing; email m.pedersen@elsevier.com.



Contributing Authors

Mohan Krishnamurthy Madwachar (OPSA, OPST) is the GM – Network Security, Almoayed Group, Bahrain. Mohan is a key contributor to their projects division and plays an important role in the organization's Network Security initiatives. Mohan comes from a strong networking, security and training background. His tenure with companies, such as Schlumberger Omnes and Secure Network Solutions India adds to his experience and expertise in implementing large and complex network and security projects.

Mohan holds leading IT industry standard and vendor certifications in systems, networking and security. He is a member of the IEEE and PMI.

Mohan would like to dedicate his contributions to this book to his brother Anand, his wife Preethi Anand and their sweet daughter Janani.

Mohan has co-authored two books *Designing & Building Enterprise DMZs* (ISBN: 1597491004) and *Configuring Juniper Networks NetScreen & SSG Firewalls* (ISBN: 1597491187) published by Syngress. He also writes in newspaper columns on various subjects and has contributed to leading content companies as a technical writer and a subject matter expert.

Eric S. Seagren (CISA, CISSP-ISSAP, SCNP, CCNA, CNE-4, MCP+I, MCSE-NT) has 10 years of experience in the computer industry, with the last eight years spent in the financial services industry working for a Fortune 100 company. Eric started his computer career working on Novell servers and performing general network troubleshooting for a small Houston-based company. Since he has been working in the financial services industry, his position and responsibilities have advanced steadily. His duties have included server administration, disaster recovery responsibilities, business continuity coordinator, Y2K remediation, network vulnerability assessment, and risk management responsibilities. He has spent the last few years as an IT architect and risk analyst, designing and evaluating secure, scalable, and redundant networks.

Eric has worked on several books as a contributing author or technical editor. These include *Hardening Network Security* (McGraw-Hill), *Hardening Network Infrastructure* (McGraw-Hill), *Hacking Exposed: Cisco Networks* (McGraw-Hill), *Configuring Check Point NGX VPN-1/FireWall-1* (Syngress), *Firewall Fundamentals* (Cisco Press), and *Designing and Building Enterprise DMZs* (Syngress). He has also received a CTM from Toastmasters of America.

Aaron W. Bayles is a senior security consultant with Sentigy, Inc. of Houston, TX. He provides service to Sentigy's clients with penetration testing, vulnerability assessment, and risk assessments for enterprise networks. He has over 9 years experience with INFOSEC, with specific experience in wireless security, penetration testing, and incident response. Aaron's background includes work as a senior security engineer with SAIC in Virginia and Texas. He is also the lead author of the Syngress book, *InfoSec Career Hacking, Sell your Skillz, Not Your Soul*.

Aaron has provided INFOSEC support and penetration testing for multiple agencies in the U.S. Department of the Treasury, such as the Financial Management Service and Securities and Exchange Commission, and the Department of Homeland Security, such as U. S. Customs and Border Protection. He holds a Bachelor's of Science degree in Computer Science with post-graduate work in Embedded Linux Programming from Sam Houston State University and is also a CISSP.

Raven Alder is a Senior Security Engineer for IOActive, a consulting firm specializing in network security design and implementation. She specializes in scalable enterprise-level security, with an emphasis on defense in depth. She designs large-scale firewall and IDS systems, and then performs vulnerability assessments and penetration tests to make sure they are performing optimally. In her copious spare time, she teaches network security for LinuxChix.org and checks cryptographic vulnerabilities for the Open Source Vulnerability Database. Raven lives in Seattle, WA. Raven was a contributor to *Nessus Network Auditing* (Syngress Publishing, ISBN: 1-931836-08-6).

Dr. Everett F. (Skip) Carter, Jr. is President of Taygeta Network Security Services (a division of Taygeta Scientific Inc.). Taygeta Scientific Inc. provides contract and consulting services in the areas of scientific computing, smart instrumentation, and specialized data analysis. Taygeta Network Security Services provides security services for real-time firewall and IDS management and monitoring, passive network traffic analysis audits, external security reviews, forensics, and incident investigation.

Skip holds a Ph.D. and an M.S. in Applied Physics from Harvard University. In addition he holds two Bachelor of Science degrees (Physics and Geophysics) from the Massachusetts Institute of Technology. Skip is a member of the American Society for Industrial Security (ASIS). He was contributing author of Syngress Publishing's book, *Hack Proofing XML* (ISBN: 1-931836-50-7). He has authored several articles for Dr. Dobbs Journal and Computer Language as well as numerous scientific papers and is a former columnist for Forth Dimensions magazine. Skip resides in Monterey, CA, with his wife, Trace, and his son, Rhett.

Josh Burke (CISSP) is an independent information security consultant in Seattle, Washington. He has held positions in networking, systems, and security over the past seven years in the technology, financial, and media sectors. A graduate of the business school at the University of Washington, Josh concentrates on balancing technical and business needs for companies in the many areas of information security. He also promotes an inclusive, positive security philosophy for companies, which encourages communicating the merits and reasons for security policies, rather than educating only on what the policies forbid.

Josh is an expert in open-source security applications such as Snort, Ethereal, and Nessus. His research interests include improving the security and resilience of the Domain Name System (DNS) and the Network Time Protocol (NTP). He also enjoys reading about the mathematics and history of cryptography, but afterward often knows less about the subject than when he started.

Eli Faskha (Security+, Check Point Certified Master Architect, CCSI, CCSE, CCSE+, MCP). Based in Panama City, Panama, Eli is Founder and President of Soluciones Seguras, a company that specializes in network security and is a Check Point Gold Partner and Nokia Authorized Partner. He was Assistant Technical Editor for Syngress' Configuring Check Point NGX VPN-1/Firewall-1 (ISBN: 1597490318) book and Contributing Author for Syngress' Building DMZs for the Enterprise (ISBN: 1597491004). Eli is the most experienced Check Point Certified Security Instructor and Nokia Instructor in the region, and has taught participants from over twenty different countries, in both English and Spanish. A 1993 graduate of the University of Pennsylvania's Wharton School and Moore School of Engineering, he also received an MBA from Georgetown University in 1995. He has more than 8 years of Internet development and networking experience, starting with web development of the largest Internet portal in Panama in 1999 and 2000, managing a Verisign affiliate in 2001, and running his own company since then. Eli has written several articles for the local media and has been recognized for his contributions to Internet development in Panama.

Contents

Chapter 1 Presenting the Business Case for Open Source Software 1
Introduction 2
The Costs of Using Free Security Solutions 2
Training Costs 2
Hardware Costs 3
Consulting Costs 3
Hidden Costs 4
The Savings of Using Free Security Solutions 5
Purchase Costs 5
Maintenance Costs 6
Customization Costs 6
Comparing Free Solutions with Commercial Solutions 7
Strengths of Free Solutions 7
Weaknesses of Free Solutions 8
Evaluating Individual Solutions 10
"Selling" a Free Solution 13
Selling by Doing 13
Presenting a Proposal 14
Summary 15
Solutions Fast Track 15
Frequently Asked Questions 16

Chapter 2 Hardening the Operating System 17
Introduction 18
Updating the Operating System 18
Red Hat Linux Errata and Update Service Packages 18
Handling Maintenance Issues 19
Red Hat Linux Errata: Fixes and Advisories 20
Bug Fix Case Study 23
Manually Disabling Unnecessary Services and Ports 25
Services to Disable 26
The xinetd.conf File 26
Locking Down Ports 28
Well-Known and Registered Ports 28
Determining Ports to Block 30
Blocking Ports 30
Stand-Alone Services 31

Hardening the System with Bastille 32
Bastille Functions 33
Bastille Versions 35
Implementing Bastille 35
Undoing Bastille Changes 41
Controlling and Auditing Root Access with Sudo 42
System Requirements 44
The Sudo Command 44
Installing Sudo 45
Configuring Sudo 47
Running Sudo 50
No Password 52
Sudo Logging 53
Managing Your Log Files 56
Using Logging Enhancers 57
SWATCH 57
Scanlogd 59
Syslogd-ng 61
Security Enhanced Linux 63
Securing Novell SUSE Linux 68
Firewall Configuration 72
Novell AppArmor 74
Host Intrusion Prevention System 77
Linux Benchmark Tools 79
Summary 84
Solutions Fast Track 85
Frequently Asked Questions 89

Chapter 3 Enumeration and Scanning Your Network 91

Introduction 92
Scanning 92
Enumeration 92
How Scanning Works 94
Port Scanning 94
Going Behind the Scenes with Enumeration 96
Service Identification 96
RPC Enumeration 97
Fingerprinting 97
Open Source Tools 98
Scanning 98
Fyodor's nmap 98
netenum: Ping Sweep 103

unicornscan: Port Scan 103
scanrand: Port Scan 104
Enumeration 106
nmap: Banner Grabbing 106
Windows Enumeration: smbgetserverinfo/smbdumpusers 112
Summary 116
Frequently Asked Questions 119

Chapter 4 Introducing Intrusion Detection and Snort 121
Introduction 122
How an IDS Works 123
What Will an IDS Do for Me? 124
What Won't an IDS Do for Me? 125
Where Snort Fits 126
Snort System Requirements 127
Hardware 127
Operating System 128
Other Software 128
Exploring Snort's Features 129
Packet Sniffer 130
Preprocessor 131
Detection Engine 132
Alerting/Logging Component 133
Using Snort on Your Network 136
Snort's Uses 138
Using Snort as a Packet Sniffer and Logger 138
Using Snort as an NIDS 143
Snort and Your Network Architecture 143
Snort and Switched Networks 147
Pitfalls When Running Snort 149
False Alerts 150
Upgrading Snort 150
Security Considerations with Snort 151
Snort Is Susceptible to Attacks 151
Securing Your Snort System 152
Summary 154
Solutions Fast Track 154
Frequently Asked Questions 156

Chapter 5 Installing and Configuring Snort and Add-Ons . . . 157
Placing Your NIDS 158
Configuring Snort on Linux 160

Configuring Snort Options 160
Using a GUI Front-End for Snort 165
Basic Analysis and Security Engine 165
Other Snort Add-Ons 172
Using Oinkmaster 173
Additional Research 174
Demonstrating Effectiveness 175
Summary 177
Solutions Fast Track 177
Frequently Asked Questions 178

Chapter 6 Advanced Snort Deployment 181
Introduction 182
Monitoring the Network 182
VLAN 182
Configuring Channel Bonding for Linux 183
Snort Rulesets 184
Plug-Ins 188
Preprocessor Plug-Ins 188
Detection Plug-Ins 195
Output Plug-Ins 196
Snort Inline 196
Solving Specific Security Requirements 197
Policy Enforcement 197
Catching Internal Policy Violators 197
Banned IP Address Watchlists 198
Network Operations Support 198
Forensics and Incident Handling 198
Summary 200
Solutions Fast Track 200
Frequently Asked Questions 202

Chapter 7 Network Analysis, Troubleshooting, and Packet Sniffing 203
Introduction 204
What Is Network Analysis and Sniffing? 204
Who Uses Network Analysis? 207
How Are Intruders Using Sniffers? 207
What Does Sniffed Data Look Like? 209
Common Network Analyzers 210
How Does It Work? 212
Explaining Ethernet 212
Understanding the Open Systems Interconnection Model . . .213

Layer 1: Physical 215
Layer 2: Data Link 215
Layer 3: Network 217
Layer 4: Transport 218
Layer 5: Session 220
Layer 6: Presentation 221
Layer 7 Application 221
CSMA/CD 223
The Major Protocols: IP, TCP, UDP, and ICMP 224
IP 224
Internet Control Message Protocol 225
TCP 225
UDP 226
Hardware: Cable Taps, Hubs, and Switches 226
Port Mirroring 228
Defeating Switches 229
Sniffing Wireless 231
Hardware Requirements 231
Software 232
Protocol Dissection 233
DNS 233
NTP 235
HTTP 236
SMTP 238
Protecting Against Sniffers 239
Network Analysis and Policy 241
Frequently Asked Questions 246

Chapter 8 Basics of Cryptography and Encryption 249

Introduction 250
Algorithms 250
What Is Encryption? 251
Symmetric Encryption Algorithms 251
Data Encryption Standard and
Triple Data Encryption Standard 252
Advanced Encryption Standard (Rijndael) 253
IDEA 254
Asymmetric Encryption Algorithms 255
Diffie-Hellman 256
El Gamal 257
RSA 258
Hashing Algorithms 258
Concepts of Using Cryptography 260

Confidentiality 261
Integrity 262
Digital Signatures 263
MITM Attacks 263
Authentication 265
Non-Repudiation 265
Access Control 265
One-time Pad 265
Summary 267
Solutions Fast Track 267
Frequently Asked Questions 269

Chapter 9 Perimeter Security, DMZs, Remote Access, and VPNs 271
Introduction 272
Firewall Types 272
Firewall Architectures 274
Screened Subnet 274
One-Legged 276
True DMZ 277
Implementing Firewalls 278
Hardware versus Software Firewalls 278
Configuring netfilter 279
Choosing a Linux Version 279
Choosing Installation Media 279
Linux Firewall Operation 282
Configuration Examples 287
GUIs 298
Smoothwall 316
Providing Secure Remote Access 325
Providing VPN Access 326
OpenSSL VPN 328
Pros 329
Cons 330
Using the X Window System 331
Summary 338
Solutions Fast Track 338
Frequently Asked Questions 340

Chapter 10 Linux Bastion Hosts 341
Introduction 342
System Installation 342
Disk Partitions 343
Choosing a Linux Version 343

Choosing Distribution Media 344
Choosing a Specific Distribution 345
Removing Optional Components 346
Minimizing Services 347
Removing Optional Software 349
Choosing a Window Manager 352
Additional Steps 353
Configure Automatic Time Synchronization 353
Patching and Updates 355
Updating Software Packages 355
Updating the Kernel 356
Removing SUID Programs 357
SELinux Policy Development 357
TCP/IP Stack Hardening 359
Automated Hardening Scripts 360
Controlling Access to Resources 362
Address-Based Access Control 362
Configuring TCP Wrappers 362
Configuring IPTables 363
Auditing Access to Resources 366
Enabling the Audit Daemon 366
Enabling the Syslog Daemon 367
Viewing and Managing the Logs 368
Configuring Swatch 368
Configuring Logwatch 369
Remote Administration 370
SSH 371
Remote GUI 372
Bastion Host Configurations 373
Configuring a Web Server 373
Configuring an FTP Server 374
Configuring an SMTP Relay Server 376
Configuring a DNS Server 377
Bastion Host Maintenance and Support 379
Linux Bastion Host Checklist 379
Summary 380
Solutions Fast Track 380
Frequently Asked Questions 382

Chapter 11 Apache Web Server Hardening. 383
Understanding Common
Vulnerabilities Within Apache Web Server 384
Poor Application Configuration 384
Unsecured Web-Based Code 384
Inherent Apache Security Flaws 384
Foundational OS Vulnerabilities 385
Patching and Securing the OS 385
Patching Unix, Linux, and BSD Operating Systems 386
Configuring a Secure Operating System 386
Hardening the Apache Application 386
Prepare the OS for Apache Web Server 387
Acquire, Compile, and Install Apache Web Server Software . .388
Verify Source Code Integrity 388
Compile the Source Code 388
Configure the httpd.conf File 392
Recommended modsecurity.conf File 393
User Directives 394
Performance/Denial-of-Service (DoS) Directives 395
Server Software Obfuscation Directives 396
Access Control Directives 396
Authentication Mechanisms 397
Directory Functionality Directives 398
Logging Directives 398
Remove Default/Unneeded Apache Files 399
Update Ownership/Permissions 400
Monitoring the Server for Secure Operation 400

Index 403

Chapter 1

Presenting the Business Case for Open Source Software

Solutions in this chapter:

- **The Costs of Using Free Solutions?**
- **The Savings of Using Free Solutions?**
- **Comparing Free Solutions with Commercial Solutions**
- **"Selling" a Free Solution**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

You may be looking for inexpensive ways to solve a security problem and want to know more about the free tools that are available. This book will guide you to some of the best free solutions for securing Red Hat Linux. In some environments, taking the initiative and implementing any type of security measures can get you in trouble; even with the best planning, problems can arise. This chapter will help you gain the support you need in order to implement a cost saving solution.

Whether you are the person implementing the changes and need to "sell" the solution to your manager, or you're the person making the decisions and need to understand the true implications of a particular "free" solution, this chapter will help you find solutions to your security problems. This chapter discusses some of the hidden costs associated with free solutions and clarifies what comes from those solutions. This chapter also addresses the fact that in most cases, an apples-to-apples comparison between a free package and a commercial product is not feasible. With all of this information, you should be in a good position to propose a solution and back up your choice with some compelling business arguments.

The Costs of Using Free Security Solutions

In the case of security solutions, few things in life are free. And while you may not pay for a security solution itself, there are costs associated with implementing a solution that are not obvious. In most cases, your security needs dictate which solutions are appropriate; if there is not a free solution available, you have to use commercial tools. Fortunately, there are a lot of high-quality free solutions available. The cross section included in subsequent chapters is aimed at providing a spectrum of solutions with a variety of sophistication levels. If you dive headlong into implementing a free solution without adequate knowledge and research, it could end up costing you more than if you had purchased a commercial solution.

Training Costs

Training costs are one of the biggest expenses when it comes to implementing a free solution. First are the direct training expenses (e.g., sending someone for classroom instruction). Your options may be limited when it comes to training for free software solutions. In most cases, training does not exist in a focused format (i.e., you probably won't find a class on netfilter firewalls). Instead, you may be able to find applicable training indirectly, such as in classes on general Linux use or administration.

Another training cost is materials (e.g., books). Aside from this book, there will likely be areas where you want more specialized information. For example, if you are implementing a Snort intrusion detection system (IDS), this book walks you through setting up Snort. A small library covering the specific software you have deployed is a worthwhile investment.

You will also incur training costs, such as not having access to an employee during training. This time away from work is an expense, because you are paying for an asset that isn't available. The same is true if the employee is on site and "self training."

Hardware Costs

A security appliance is a device that doesn't require a computer and is only used for its intended purpose, while all of the free solutions require a system to run on. Luckily, the requirements are usually minimal; therefore, you can often use an old PC. Also, some of the software can be easily stacked on the same system. In other cases, the physical location required for the software (e.g., sniffers, IDSes, or traffic reporting tools) can make a system unsafe. Rarely does a system require enough resources to make using the same host for any other function impractical (e.g., the Snort IDS logging capability can quickly eat up disk space, leaving little to no resources for other programs).

If there are no old systems available, there are many online retailers offering older systems at affordable rates. A large portion of the cost for low-end PC's is often for the operating system. Many retailers offer affordable systems that either include Linux as the operating system, or come without an operating system installed. These allow you to purchase a relatively modern system cheaply, and then install your own OS on it. This can be a viable option for running security tools and providing user workstations.

Consulting Costs

You must carefully weigh and balance where you spend your money. Too little training and you will end up hiring consultants. Implementing, configuring, or fixing your free firewall can cost a lot, more than if you had bought a firewall. With small commercial firewalls costing around $500.00, it doesn't take long before free isn't so free.

With that said, don't be afraid to call a consultant if necessary. Having a well-paid consultant configure your free solution and make sure that it's implemented using best practices is a steal compared to implementing some proprietary solutions. A consultant can also act as a trainer. You can shadow the consultant and see how and what is being done, and you can ask questions and learn why things are done a certain way. In this way you can have your solution set up by someone who is knowledgeable and experienced, and provide training and guidance to the in-house personnel.

If you have ever had to rely on consultants, you probably know they are not always a "good buy." Sometimes they are not as knowledgeable as you were led to believe. The key is to communicate with the consulting firm, being very clear about what your needs are. A good consultant can save the day.



WARNING

You should always be careful when cutting consulting budgets. I have seen attempts to save money end up costing more. In almost all cases, getting a consultant in quickly is the best course of action and the most cost effective in the long run. If you find a skilled consultant you like, a monthly retainer might be a good investment.

Hidden Costs

What are all the costs of a free solution? For starters, power consumption. I had a Windows 98 system that was only being used as a print server. It occurred to me that the PC cost me approximately $7 per month in electricity. With a dedicated print server costing only about $30.00 and using virtually no electricity, I would save money within five months by buying the print server. The Pentium II running Windows 98 was technically "free," but paying for electricity to keep it running was not the most cost effective choice. Some security tools are not offered as a commercial appliance and some are (e.g., small, low cost firewalls that use far less power than a standard desktop PC are available from several manufacturers). Your cost for electricity will vary. Based on your electric bill, you can calculate with a high degree of accuracy what a given device costs.

Another consideration is heating, ventilation, and air conditioning (HVAC) costs. HVAC is basically the climate controls. Additional computers create additional heat, which costs more money for air conditioning. The same considerations apply as for power consumption. If a stand-alone appliance is not an option, the additional HVAC requirements are an unavoidable cost; however, in those cases where a more efficient application exists, they almost always produce less heat than a normal workstation. This also applies to the difference between an older computer and a newer computer. Newer systems that demand more power and cooling when they are being heavily utilized, often incorporate superior energy-saving characteristics than the older systems.

There is also the cost of real estate. A decommissioned full-sized tower PC takes up a lot more space than a new commercial appliance the size of a cigar box. You may have plenty of room now, but as the server room gets more and more crowded, space could become an issue. A keyboard, video, and mouse (KVM) switch might save more in space than it costs to buy. As the servers become increasingly tightly packed, good air flow and adequate cooling will be inhibited, and physical access to the systems for operation or maintenance will also be difficult.

Inefficiency is another cost of free solutions with respect to the fact that the support staff are likely unfamiliar with the new free solutions. When a staff member performs a task on a

new firewall, it takes longer to do than if they are familiar with the firewall. This inefficiency costs the time to complete a task; however, if an outage or business disruption occurs, this delay could result in lost profit or business. These delays must also be accounted for when planning projects and other activities.

Free solutions are usually produced by small organizations or by an individual. These solutions may do an excellent job in their assigned roles, but may not be well known. This could be a liability if the individual who configured your free solution leaves or is otherwise unavailable. If you have a PIX firewall that needs work, you probably would not have a hard time locating a resource. On the other hand, if you need someone to take over the administration of an obscure free solution, finding someone could be difficult. This difficulty could manifest itself as a hidden cost by increasing the delay before a problem can be addressed, having to pay a premium for a consultant, or any number of other inefficiencies.

The Savings of Using Free Security Solutions

The following section discusses how a free security solution can save you money. The primary savings is obvious: you didn't pay for the product; however, there are additional benefits. This section offers a detailed look into the benefits of using free software. By evaluating the expected savings and costs, you can form a more practical, accurate picture of what will be gained by implementing a free security solution.

Purchase Costs

The purchase cost is one of the single largest cost savings of using free software. The best example of this is with firewalls. A small Linksys or Netgear firewall costs around $20.00 to $50.00. They use almost no power, support port forwarding, perform Network Address Translation (NAT), act as a Dynamic Host Configuration Protocol (DHCP) server, and are stateful packet filters. Suppose you use Linux and netfilter to run a firewall for free. Odds are it will cost more to pay for the employee's time to set up the Linux firewall than the Linksys would cost to buy. Firewalls are one of the best examples of how readily available affordable commercial solutions can be.

You can still save money on purchases. Some types of products, particularly IDSes, network analysis and reporting tools, and commercial Virtual Private Network (VPN) solutions can cost staggering amounts of money. When comparing prices, come as close as possible to comparing like products. Using the most expensive "deluxe" software suite available as the price for decision making is misleading. The free solution will not have the same features and capabilities as the commercial version. Look at the features you think you need as a starting point for which commercial products would be viable options. Use the costs of those products as your basis for determining what the free solution will save you.

Maintenance Costs

Maintenance can be expensive; it is not uncommon for a yearly maintenance contract to cost 10 percent of the purchase price. This price will also fluctuate, as almost all vendors have various support tiers with varying response times and service level agreements (SLAs). The reality is, however, if you opt for the free solution and spend the 10 percent on training instead, you would probably have a very high level of responsiveness from your own in-house staff. Ensuring an equivalent level of responsiveness and availability from the vendor would likely cost you a large sum. Your own support staff could probably go to the office or address the issue remotely far more quickly than all but the largest and most well-established vendors. Even if a vendor can have someone on site in two hours, sometimes getting a live person to return your call and schedule the emergency appointment takes time. You can probably reach your own staff as quickly, if not more so. The level of service you expect should be factored in when estimating the cost savings available by not having to purchase a maintenance contract.

Customization Costs

Customization is an area that can offer huge gains or be inconsequential, depending on your circumstances. If you purchase a commercial product, you may find that there is no way it can be customized for your environment. If some degree of customization is available, it is rarely free. Often, the hourly rate for such services is at a premium, the assumption being you must really want or need the desired functionality if you are willing to pay to add it. With some free solutions, this customization can be affordable, or even free, if you have the expertise. However, not all free software is customizable. Just because it's free does not always mean it is open source. Open source software is software where the source code (i.e., the programming code used to make it run) is freely available. When software is open source, you can download the source code and edit it to your heart's content. You can add as few or as many custom features as you want.

Obviously, this is an advantage that not everyone will need or have the means to take advantage of. Depending on the software package in question, some are programmed using different programming languages, so even if you have a resource who knows enough to be able to customize the program, they might not know the particular programming language that is required. Customization is also something you don't know you need until you are well into the implementation phase. If you know your customization needs ahead of time you can investigate and weigh the costs accordingly. Generally speaking, even if the cost is the same to customize the free solution as a comparable commercial solution, the level of customization that is possible is often (but not always) equivalent or better with the free solution.

Comparing Free Solutions with Commercial Solutions

When it comes to making an informed decision as to whether to purchase a commercial solution or implement a free solution, there are some additional non-dollar-related considerations to take into account. First and foremost, compare like functionality. Don't compare the deluxe version of the commercial product to the free version; they won't have the same features or learning curve, or require the same hardware. Ultimately, by making the most informed and well-reasoned comparison possible, the best solution will be chosen.

Strengths of Free Solutions

One advantage free solutions often have over their commercial counterparts is that of development speed. This varies from one product to another; not all free products have quick development cycles. The open-source packages often have very fast development cycles and can address the latest security issue more quickly than their commercial counterparts. If you want to stay on the cutting edge, free software (especially open-source software) might be a better path than commercial solutions.

Previously, we discussed customization as a cost savings with some free software. This is because often you can do the customizing yourself instead of paying the vendor to do it for you. Customization is worth mentioning as a strength of its own, above and beyond the cost savings. Again, not all free software is customizable. Sometimes the best software in a particular category uses closed code and there is no way for you to perform any customization. But one of the greatest strengths of the open-source movement is that anyone and everyone has the freedom to edit, customize, and improve the software.

A potential strength of free solutions is the speed with which they can be implemented (which is different than the development speed). When I speak of the implementation speed of free software I am referring to the time it takes to get the software loaded and working. This includes not only installation, but also the red tape sometimes involved in making significant purchases. For example, suppose you are trying to form a business partnership that will be beneficial to your organization. The nature of the arrangement is such that time is of the essence; the sooner the partnership is completed the better. The partnership involves network connectivity to facilitate the exchange of information. After reviewing the plans of how it would be done, your potential partner is hesitant to go through with it, because you lack adequate firewall protection. Maybe your current Internet connection is filtered with a consumer-level home router/firewall and you need a separate demilitarized zone (DMZ) with some advanced NATing rules and better logging. You could contact a vendor, wait for a response, get a quote on the price, and pass that to your manager for approval. After your manager approves the purchase, you hand it to accounting and they make the purchase and arrange shipping. Once it arrives, you must install and configure the new firewall and then

test it. A faster approach would be to grab the old PC from the closet, download and install Linux on it, and configure the firewall. If your environment allows it, implementing the free solution could be much faster. In environments where there are restrictions on permitted vendors, permitted software, permitted hardware, and so on, getting approval for a free solution could be more difficult and time consuming than a commercial solution. Ultimately, your environment will dictate whether implementation speed can truly pan out as an advantage or not.

You might think that all free software is produced by some kid after school and will be unstable and lacking the quality control of a commercial software development project. While this is certainly true some of the time, at other times it could not be farther from the truth. The fact is that the larger, well-established open-sourced projects can have hundreds of programmers reviewing, revising, scrutinizing, and modifying the code. Very few commercial companies have the same amount of resources to put into a single software product. This means that in many cases you are getting software that has been through more peer review and testing than the commercial equivalent. This is not always true; in many cases the free software has very little quality control and you, as the user, are really doing the testing. Basically, this means that the quality of free solutions will have a lot of variance. To increase the odds that you are not trying to implement buggy software, do your homework. If you stick to mature products that have a proven track record you will certainly improve your odds. Avoiding new releases that implement major architectural changes may help as well. If the current release of a product you are using incorporates newly added support for the latest chipset, it might be wise to wait for that release to be tested a little more before deploying it in your environment. For an excellent and lengthy article on the merits of free software, refer to http://www.dwheeler.com/oss_fs_why.html. In reality, some of the free offerings are not fit to be run in any sort of critical role, while others can do so with aplomb. Ultimately, not all free software is "cheap" software; some of the free offerings are of very high technical quality.

Weaknesses of Free Solutions

The single biggest drawback to implementing a free solution in a production environment is one of support, or lack of support. When you download something for free from the Internet, there is generally no phone number to call and ask questions. This is sometimes mitigated by high quality documentation, and in some cases extensive online user forums where you can ask questions and receive help from the creator of the package or other users. On the other hand, high-quality documentation is the exception rather than the norm, and many of the free utilities have little in the way of documentation. This consideration is one of the biggest concerns for management. Generally speaking, the more mission critical the role of the security software is, the more hesitant you should be about implementing a solution with minimal support. If you are a company that depends on the Internet, you should require a

higher level expertise from in-house technical staff before implementing a free Linux firewall, compared with another company that makes money in a storefront and only uses the Internet to surf the Web. This isn't to say that the support cannot be adequate with free software or that you shouldn't use free solutions to fulfill critical needs, only that you need to do so knowingly and after careful consideration and planning.

The management capabilities of free software solutions are typically not as robust as they are with commercial offerings. Your particular product will determine if this is a real consideration or not. Most often the presence or absence of management capabilities is more noticeable with free IDSes, antivirus, and antispyware offerings. The common denominator here is that these products require frequent updates in order to maintain their value and do their job effectively. An enterprise class antivirus program will offer a lot of control and features around signature updates, such as when and how to perform the updates and how to handle things when a virus is detected. The free solutions are generally more limited, often requiring the scanning or updating process to be performed manually, and responding to a positive detection may have to be an interactive process, rather than an automated one.

Another area where the free solutions are also sometimes lacking is reporting. While some offer excellent reporting, many others offer little to no reporting capability. In most cases, you will be able to manually configure some type of reporting on your own using freely available utilities. Even if you can arrange for some automated logging or reporting to be generated, it won't be as simple or quick as it would be if it were a commercial product that supported that functionality natively. As you begin considering free solutions, you will want to also consider not only the logging capabilities you *want*, but those you *need*. In many cases, if you are in a highly regulated industry, such as banking, or healthcare, the lack of adequate logging capability is the determining factor that leads to a decision to go with commercial software. If you have auditors you need to satisfy, you will want to research the audit trail you will be able to generate carefully, before coming to a strategic decision on your solution.

Previously, we touched on the fact that the free solutions are often not well known, and how this can translate into a hidden cost in consulting fees. This liability can go beyond consulting fees. If you were hiring a new employee and specified that they need to know Cisco equipment, you could undoubtedly find someone in short order. If you specified you wanted them to be familiar with some little-known free solution you have implemented, you could have a very hard time finding someone. That's not to say that they couldn't be trained, but again, there are costs and disadvantages associated with that. The familiarity (or lack thereof) could also cause the time it takes to implement a solution to be longer than with a more widely understood technology. Speed of implementation was mentioned as a potential asset, but it can easily be a liability if there is no one available who understands the solution. Ultimately, there are advantages to using industry standard solutions over less widely deployed offerings.

Evaluating Individual Solutions

As you do your research, you will need to determine if the free solution is the best solution. There are a whole host of factors which will go into making this determination. The following list briefly summarizes the steps needed to make a determination as to whether or not a free solution is the best solution for you.

1. **Identify Your Options** This can be the hardest part of the process, knowing what free alternatives exist. Hopefully this book will help, but there are also on-line sites to help you find free software. One of the largest sites housing open source software is http://sourceforge.net/index.php. Also check out http://freshmeat.net/. You can find a more programmer-oriented site containing only software that runs on Linux at www.icewalkers.com/. A directory of free software is located at http://directory.fsf.org/. A similar directory of free software for Microsoft Windows is located at http://osswin.sourceforge.net/. Finally, a CD containing some "top picks" of free software for use on Windows is located at www.theopencd.org/.
2. **Research Each Option** Typically, this will mean doing searches on the software. Take note of how many problems people have, and if they have been fixed. Check the developer's Web site and documentation. See if the documentation is well-crafted and complete. This is when you will weed out the majority of candidates and hopefully be left with a list of quality choices.
3. **Compare Products** The previous step is meant to sort out the best free solutions. This step is aimed at comparing the best free solutions against their commercial counterparts. This is where you may rule out some products as too expensive or to hard to use. Metrics to use for comparison include:
    - **Functionality** The product must meet your business needs to be considered. Pay attention to volumes. The product might do what you want, but not on the scale you want it to. Consider if the product will work with other utilities or if it uses proprietary and closed source methods, protocols, or algorithms. These traits may act as limiters and hinder flexibility later on.
    - **Cost** This is one of the major reasons you are considering a free solution. Try and be as accurate as possible in your estimates of the true costs, including things such as purchase cost, maintenance, training, upgrades, and so on.
    - **Momentum** How well established is the product? Remember this is a consideration for free software and commercial software. The more well established the software is, the better the odds the creators will be around in the future. A larger more well-established project will also likely have better community support and reliability. Included in the overall momentum is to

look at how active the project is. You don't want to invest a lot of time and energy in a product that is likely to just die off and fade away.

- **Support** What does support cost? Is it available? How timely is the support? What format does support take (online forums, e-mail, phone, and so on).
- **Performance** Which solutions are the best performers? This includes speed, efficiency, and reliability. A powerful software package that crashes every hour isn't a viable option.
- **Usability** Is the product use friendly? If the learning curve is very high, your training costs will rise. If the product doesn't have a feature or function you like, can you customize it and make it more user friendly?
- **Security** Even for a security tool you must consider the security implications. Is the product secure? Will it be handling secure data? Are you opening up any new security risks? What type of auditing and logging can it produce?
- **Legal and License Issues** Be sure and review the license agreement closely. Many times the free software is not free if you are a business, or there are special restrictions on the number of installations or other criteria. When in doubt have your legal counsel review the license agreement for you.
- **Individual Criteria** These are any special needs or requirements unique to your environment. What's good for other organizations might not work for you.

4. **Perform Detailed Testing** At this stage, you have hopefully narrowed the playing field down to just one or two selections. It's time to put them through a real test and see if they do what they claim they to do. This can be done in a lab or possibly on the production network, depending on the risks involved and the nature of the product. You will need to evaluate how best to perform your detailed testing based on your circumstances.
5. **Come to a Conclusion** After all this research, you can make a decision on what you think the best solution is. Whether you are the final approver or you need to forward your recommendation to someone else for approval, at this point you should have all the facts collected in order for a good decision to be made.

Remember, the preceding steps leave a lot of room for flexibility. They may be performed in a more or less structured fashion. You might not formally cover all the steps, but in one form or another those steps should occur. The more thoroughly you document the steps, the more you will be in a better position to justify your choices.

Now that we have discussed the many ways that the cost of a free solution may be higher or lower than the commercial equivalent, let's look at an example. Suppose your

manager wants you to provide a reporting mechanism to see who is using the majority of the Internet bandwidth. Your manager also wants to know what the user(s) in question are using the bandwidth for. You search around and learn about a product called *nGenius Performance Manager*, which is made by Netscout (www.netscout.com/products/pm_home.asp). According to your research, it will do what you want and more. The graphs and charts it can produce are excellent, and it provides an extremely granular look into the traffic flowing across your network. In the free department, you've looked at *ntop* and it seems pretty neat, not as granular, but still offering a respectable amount of data and reporting for free. You call up netscout and get some list pricing for the nGenius equipment. The server licenses have a scaled price structure according to the software's capabilities, so you inquire about the most economic server license they offer, which is $20,000.00 list. You will need at least a single probe to sniff and collect data, which is another $5,000.00. You will need to run this on a server and the old one probably won't work, so there's another $2,000.00. The yearly maintenance contract will be 10 percent of the purchase price, meaning another $2,500.00, bringing the grand total to $29,500.00, less any price breaks from list you might get.

If you then went to your boss and used the $30,000.00 price tag to justify why you should implement a free traffic reporting and analysis tool, your presentation wouldn't be telling the whole story. First off, none of the free products come close to the power and functionality of nGenius, so you are not comparing like products. There are other less expensive alternatives, which would represent a much more accurate comparison to use as a cost savings example. Second, even if money were no object, deploying an enterprise-class solution like nGenius is probably not the best choice. Along with the impressive array of features comes a fairly steep learning curve. After implementing such a solution, your in-house staff might have more difficulty learning how to use it than they would with one of the free (and simpler) solutions. Third, you may not need the level of detail and sophistication that nGenius offers. If *ntop* or a comparable free solution can offer all the reporting and metrics that you are looking for, deploying a more complex solution may not be wise. *ntop* may be the best choice for your organization, but presenting that choice as a $30,000.00 cost savings is far from accurate.

nGenius is the Cadillac of network analysis tools. It has a staggering array of features and an impressive level of customization you can perform without getting into actual programming. If I had the budget and the need, it is the product I would use. That being said, is *ntop* just as powerful? Not even close. But, in a small organization, the added features *nGenius* has to offer would likely never be used. With a price tag of free, *ntop* or one of the many other free alternatives is likely to do everything that is needed, and with a lot smaller learning curve.

"Selling" a Free Solution

If you are in a position where you can implement a new security solution without having to receive anyone else's approval, you probably don't need to read this section. If on the other hand, you have to get someone to sign off on your plan, this should be helpful. If you do need approval, you are basically gong to try and "sell" your solution, much like a salesman, highlighting the benefits, and realistically noting any disadvantages to your proposed solution. Remember, the objective of presenting a solution is not to "win" by getting to do things your way. The objective is to provide the decision makers with the most complete and accurate information so that the best decision can be made. Your own judgment of the environment and your target audience will play a large part in what constitutes the best approach for you to take. Hopefully, some helpful guidelines as to how to approach gaining approval can help improve your odds of success.

Selling by Doing

One method of demonstrating the power and effectiveness of a given solution is to actually demonstrate the solution. If the environment allows, and you have the resources, it might be feasible to install the software in question, generate the reports, and present the facts, along with a demonstration of what the software can do. You don't want to do anything that is inappropriate; if the change control procedures don't allow such spontaneity, you will need to revise your approach. Assuming you have the freedom to do so, saying the software generates graphs and reports and traffic usage, broken up by protocol and the computer in question, rarely has the same impact as seeing that same graph. Not only does it provide factual real evidence of the utilities value, it also demonstrates your initiative and forethought.



WARNING

Let's be perfectly clear here, I'm not advocating that you go and implement some solution without proper management approval when policy says you shouldn't. You need to evaluate your environment and factor in things such as climate, policy, risks, benefits, and so on, to determine if it's wise to implement something without getting all the proper approvals ahead of time. Again, in some environments this would be perfectly okay and your manager would be elated at your ingenuity and initiative, while in others you could end up looking for a new job. As always, exercise good judgment and when in doubt, take the conservative approach.

Presenting a Proposal

If you do not have the luxury of implementing something and then asking for "approval," you will need to create a proposal with all of the relevant information. You can certainly do both, including the sample data from the utility in the proposal. The truth is, "presenting a proposal" sounds very formal, and it can be, but it doesn't have to be. Some organizations have much more formal procedures in place than others. Presenting your proposal may be as structured as using a standardized template with forms to fill out and submit, and meetings with PowerPoint presentations. It could also just as easily mean talking with your manager over lunch and telling him what you would like to do.

Regardless of the format you employ for your proposal, there are certain common elements you will want to touch upon, verbally or on paper. If you address all these issues as accurately as possible, the odds of your venture being a success should be greatly improved. At a minimum, try and have information and answers covering the following areas concerning your proposed solution;

- What will it take? How much will it cost? How long will it take to implement? How much training will be required and of what type? How much will the training cost, and how long will it take? What hardware might be needed and what will it cost? Will it impact the user experience? If so, how?
- What will it do? What are the real capabilities, not just sales hype? Sampling actual samples from your environment, or if you can find something online, would go a long way here. Hard data is always better than a sales blurb. What are the technical limitations?
- **Assumptions** What other factors must be in place for this to work as planned? Will you need assistance with the implementation? Will an outside consultant be needed?
- **Caveats** What are the drawbacks? What makes your solution less attractive? What are potential problems that might arise?

Summary

Not all facets of implementing free security solutions are free. There are always costs of one type or another, which vary in magnitude and relevance based on your individual circumstances. Ultimately, you don't want to be yet another person who fell victim to the open source or freeware hype. These are the people who read or heard about a "free" product and rushed to implement it without doing adequate research, thus ending up with a mess that is expensive to make work or to clean up. With the proper research and planning, free solutions can provide you with some very powerful security solutions without spending a lot of resources. The real value lies in finding free software that is the simplest solution available that can still meet your needs.

Solutions Fast Track

The Costs of Using Free Security Solutions

- ☑ Training costs can quickly skyrocket, especially for classroom-based training.
- ☑ Consulting costs are not always something to be avoided. At times they can provide a very efficient way to implement a given solution while at the same time providing some sorely needed training and documentation.
- ☑ Intangibles can also add up. While items such as HVAC, power costs, and space requirements are not likely to break the bank, these are still considerations you should be aware of in order to make informed decisions.

The Savings of Using Free Security Solutions

- ☑ The biggest savings is that there are no software costs.
- ☑ No maintenance costs

Comparing Free Solutions with Commercial Solutions

- ☑ You can usually implement a free solution quicker than a commercial product, based on the time it takes to make and receive the purchase.
- ☑ A free solution's primary weakness is support. Without a toll-free number to call, you are left to educate yourself or pay someone with the appropriate skills to assist. The often sparse or non-existent documentation can sometimes be a major hindrance to a successful implementation.

- ☑ Many of the free solutions are also open source, allowing you unequaled flexibility to customize, alter, change, or even rewrite the software in question.

"Selling" a Free Solution

- ☑ Be informed of the pros and cons of the solution, and be honest about your data. Remember that it's not a contest to implement a particular solution, but rather the objective is to be well informed so that the best solution can be chosen.
- ☑ Real life examples are always better than theory. A sample graph of data from your current network (policy allowing) is always going to drive home the point better than a bullet that says the product will produce the graph.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How do I know when I have found the best solution?

A: The solution that is "best" today, might not be tomorrow. The selection of free software is rapidly changing. While there are certain leaders who will likely continue to be top picks for the foreseeable future, many other free solutions will come and go. The only way to make a good decision is to "do your homework," and if possible, consult an expert in the area you are interested in.

Q: If some of these free tools are so good, why doesn't everyone use them?

A: In the case of a large organization, the features or functionality the free solutions lack are vital, so a commercial solution may be the only option for some. For smaller organizations for whom a free solution can satisfy their needs, it typically comes down to not knowing what the options are. No one is paying to advertise free products in computer magazines, so generally only the more experienced and knowledgeable Information Technology (IT) people know about all the available products.

Q: Is free software really free?

A: Not in every sense of the word. While the software itself may cost nothing, you have to consider the costs of the hardware required to run the software, the training required to implement the software, and the potential maintenance costs (in terms of manhours and actual dollars) when considering a free solution. After adding all of this up, some free solutions can be very "cost effective," even if not truly free.

Chapter 2

Hardening the Operating System

Solutions in this chapter:

- **Updating the Operating System**
- **Handling Maintenance Issues**
- **Manually Disabling Unnecessary Services and Ports**
- **Locking Down Ports**
- **Hardening the System with Bastille**
- **Controlling and Auditing Root Access with Sudo**
- **Managing Your Log Files**
- **Using Logging Enhancers**
- **Security Enhanced Linux**
- **Securing Novell SUSE Linux**
- **Novell AppArmor**
- **Host Intrusion Prevention System**
- **Linux Benchmark Tools**

Introduction

Linux is capable of high-end security; however, the out-of-the-box configurations must be altered to meet the security needs of most businesses with an Internet presence. This chapter shows you the steps for securing a Linux system—called *hardening* the server—using both manual methods and open source security solutions. The hardening process focuses on the operating system, and is important regardless of the services offered by the server. The steps will vary slightly between services, such as e-mail and Hypertext Transfer Protocol (HTTP), but are essential for protecting any server that is connected to a network, especially the Internet. Hardening the operating system allows the server to operate efficiently and securely.

This chapter includes the essential steps an administrator must follow to harden a Unix system; specifically, a Red Hat Linux system. These steps include updating the system, disabling unnecessary services, locking down ports, logging, and maintenance. Later in this chapter you may find some information for Novell SUSE Linux. Open source programs allow administrators to automate these processes using Bastille, sudo, logging enhancers such as SWATCH, and antivirus software. Before you implement these programs, you should first understand how to harden a system manually.

Updating the Operating System

An operating system may contain many security vulnerabilities and software bugs when it is first released. Vendors, such as Red Hat, provide updates to the operating system to fix these vulnerabilities and bugs. In fact, many consulting firms recommend that companies do not purchase and implement new operating systems until the first update is available. In most cases, the first update will fix many of the problems encountered with the first release of the operating system. In this section, you will learn where to find the most current Red Hat Linux errata and updates.

Red Hat Linux Errata and Update Service Packages

The first step in hardening a Linux server is to apply the most current errata and Update Service Package to the operating system. The Update Service Package provides the latest fixes and additions to the operating system. It is a collection of fixes, corrections, and updates to the Red Hat products, such as bug fixes, security advisories, package enhancements, and add-on software. Updates can be downloaded individually as errata, but it is a good idea to start with the latest Update Service Package, and then install errata as necessary. However, you must pay to receive the Update Service Packages, and the errata are free. Many errata and Update Service Packages are not required upgrades. You need to read the documentation to determine if you need to install it.

The Update Service Packages include all of the errata in one package to keep your system up to date. After you pay for the service, you can download them directly from the Red Hat Web site. To find out more about the Update Service Packages, visit the secure site www.redhat.com/apps/support/.

You may also launch the **Software Updater** from Applications | **System Tools** | **Software Updater** from the taskbar (Red Hat Enterprise Linux 5). You have to register yourselves with RHN (Red Hat Network) and send the hardware and software profile for Red Hat to recommend appropriate updates for your system. Figure 2.1 shows the registration process through Software Updater.

Figure 2.1 Software Updater



Handling Maintenance Issues

You should apply the latest service pack and updates before the server goes live, and constantly maintain the server after it is deployed to make sure the most current required patches are installed. The more time an operating system is available to the public, the more time malicious hackers have to exploit discovered vulnerabilities. Vendors offer patches to fix these vulnerabilities as quickly as possible; in some cases, the fixes are available at the vendor's site the same day.

Administrators must also regularly test their systems using security analyzer software. Security analyzer software scans systems to uncover security vulnerabilities, and recommends fixes to close the security hole.

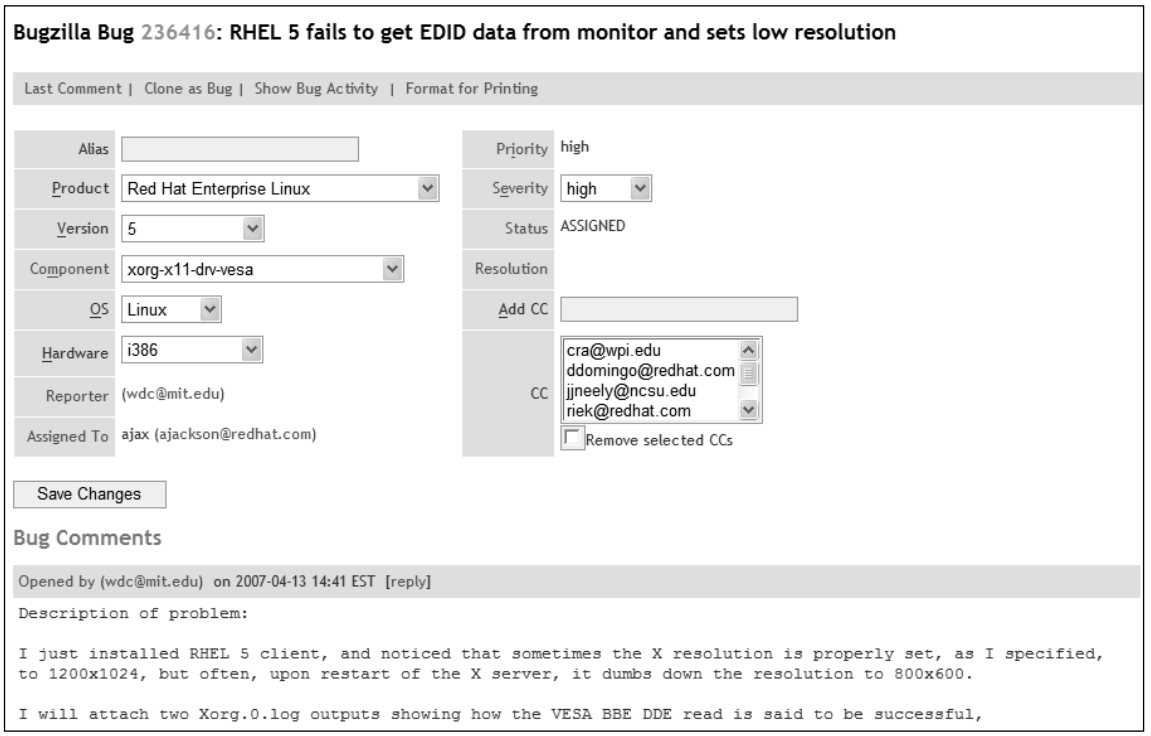
This section discusses the maintenance required to ensure that your systems are safe from the daily threats of the Internet.

Red Hat Linux Errata: Fixes and Advisories

Once your Red Hat system is live, you must make sure that the most current required Red Hat errata are installed. These errata include bug fixes, corrections, and updates to Red Hat products. You should always check the Red Hat site at www.redhat.com/apps/support for the latest errata news. The following list defines the different types of errata found at the Red Hat Updates and Errata site.

- **Bug fixes** Address coding errors discovered after the release of the product, and may be critical to program functionality. These Red Hat Package Manager tools (RPMs) can be downloaded for free. Bug fixes provide a fix to specific issues, such as a certain error message that may occur when completing an operating system task. Bug fixes should only be installed if your system experiences a specific problem. Another helpful resource is Bugzilla, the Red Hat bug-tracking system at https://bugzilla.redhat.com/. You may report a bug that you have encountered in your system through Bugzilla. Figure 2-2 shows one such notification of a bug by a user.
- **Security advisories** Provide updates that eliminate security vulnerabilities on the system. Red Hat recommends that all administrators download and install the security upgrades to avoid denial-of-service (DoS) and intrusion attacks that can result from these weaknesses. For example, a security update can be downloaded for a vulnerability that caused a memory overflow due to improper input verification in Netscape's Joint Photographic Experts Group (JPEG) code. Security updates are located at http://www.redhat.com/security/updates/
- **Package enhancements** Provide updates to the functions and features of the operating system or specific applications. Package enhancements are usually not critical to the system's integrity; they often fix functionality programs, such as an RPM that provides new features.

Figure 2.2 Notification of a Bug through Bugzilla



You also have an option of sending the bug through the **Bug Reporting Tool**. This pops-up automatically when you encounter an error during your routine work on your system. Figure 2.3 shows the Bug Reporting tool.

If you click on Show details you may find the information shown below (partial output shown here). This information is based on the nature of the bug, software and hardware configuration, and will vary from system to system. Though you may not be able to make out all that is captured by the bug reporting tool, experts in the Red Hat support will be able decode the same and work on the fixes.

Figure 2.3 Bug Reporting Tool




```
Distribution: Red Hat Enterprise Linux Server release 5 (Tikanga)
Gnome Release: 2.16.0 2006-09-04 (Red Hat, Inc)
BugBuddy Version: 2.16.0

Memory status: size: 147779584 vsize: 0 resident: 147779584 share: 0 rss: 68427776
rss_rlim: 0
CPU usage: start_time: 1189756814 rtime: 0 utime: 2224 stime: 0 cutime:2027 cstime:
0 timeout: 197 it_real_value: 0 frequency: 93

Backtrace was generated from '/usr/bin/yelp'

(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".
(no debugging symbols found)
 [Thread debugging using libthread_db enabled]
[New Thread -1208363296 (LWP 3961)]
[New Thread -1255404656 (LWP 4181)]
[New Thread -1243546736 (LWP 3963)]
[New Thread -1210463344 (LWP 3962)]
(no debugging symbols found)
 (no debugging symbols found)
```

```
0x002ae402 in __kernel_vsyscall ()
#0  0x002ae402 in __kernel_vsyscall ()
#1  0x0033dc5b in __waitpid_nocancel () from /lib/libpthread.so.0
#2  0x051d1c26 in gnome_gtk_module_info_get () from /usr/lib/libgnomeui-2.so.0
#3  <signal handler called>
. . . . . .
#48 0x08051811 in g_cclosure_marshal_VOID__VOID ()

Thread 4 (Thread -1210463344 (LWP 3962)):
#0  0x002ae402 in __kernel_vsyscall ()
No symbol table info available.
#1  0x0090a5b3 in poll () from /lib/libc.so.6
No symbol table info available.
. . . . .
#8  0x0091414e in clone () from /lib/libc.so.6
No symbol table info available.

Thread 2 (Thread -1255404656 (LWP 4181)):
#0  0x002ae402 in __kernel_vsyscall ()
No symbol table info available.
#1  0x0033a3cc in pthread_cond_timedwait@@GLIBC_2.3.2 ()
   from /lib/libpthread.so.0
. . . . .
#48 0x08051811 in g_cclosure_marshal_VOID__VOID ()
No symbol table info available.
#0  0x002ae402 in __kernel_vsyscall ()
```

Bug Fix Case Study

Once you register your system with Red Hat Network, time-to-time you may receive emails with a subject 'RHN Errata Alert'. These alerts are specific to the system you registered consisting summary of the problem, a detailed description and the actions recommended to resolve the problem.

In this case study the following mail received from Red Hat provides the details of 'kernel security update' required by the registered system (partial output shown):

```
Red Hat Network has determined that the following advisory is applicable to one or
more of the systems you have registered:

Complete information about this errata can be found at the following location:
     https://rhn.redhat.com/rhn/errata/details/Details.do?eid=5984

Security Advisory - RHSA-2007:0705-2
--------------------------------------------------------------------------------
Summary:
Important: kernel security update

Updated kernel packages that fix various security issues in the Red Hat Enterprise
Linux 5 kernel are now available.
```

```
This update has been rated as having important security impact by the Red Hat
Security Response Team.

Description:
The Linux kernel handles the basic functions of the operating system.

These new kernel packages contain fixes for the following security issues:

* a flaw in the DRM driver for Intel graphics cards that allowed a local user to
access any part of the main memory. To access the DRM functionality a user must
have access to the X server which is granted through the graphical login. This also
only affected systems with an Intel 965 or later graphic chipset. (CVE-2007-3851,
Important)

* a flaw in the VFAT compat ioctl handling on 64-bit systems that allowed a local
user to corrupt a kernel_dirent struct and cause a denial of service (system
crash). (CVE-2007-2878, Important)

. . . . . (output truncated)

Red Hat Enterprise Linux 5 users are advised to upgrade to these packages, which
contain backported patches to correct these issues.

References:
http://www.redhat.com/security/updates/classification/#important
--------------------------------------------------------------------------------

-------------
Taking Action
-------------
You may address the issues outlined in this advisory in two ways:

     - select your server name by clicking on its name from the list
       available at the following location, and then schedule an
       errata update for it:
           https://rhn.redhat.com/rhn/systems/SystemList.do

     - run the Update Agent on each affected server.


. . . . (output truncated)
---------------------
Affected Systems List
---------------------
This Errata Advisory may apply to the systems listed below. If you know that this
errata does not apply to a system listed, it might be possible that the package
profile for that server is out of date. In that case you should run 'up2date -p' as
root on the system in question to refresh your software profile.

There is 1 affected system registered in 'Your RHN' (only systems for which you
```

```
have explicitly enabled Errata Alerts are shown).

Release   Arch       Profile Name
--------  --------   ------------
5Server   i686       linux11

The Red Hat Network Team
```

As you may notice from the above mail the registered system requires a kernel security update. Now you need to follow the steps outlined under 'Taking Action' section to ensure your system is updated. In this case this advisory recommends you schedule errata update and run the Update Agent on the affected server.

Manually Disabling Unnecessary Services and Ports

As a Linux administrator or a security administrator it is essential for you to define the following:

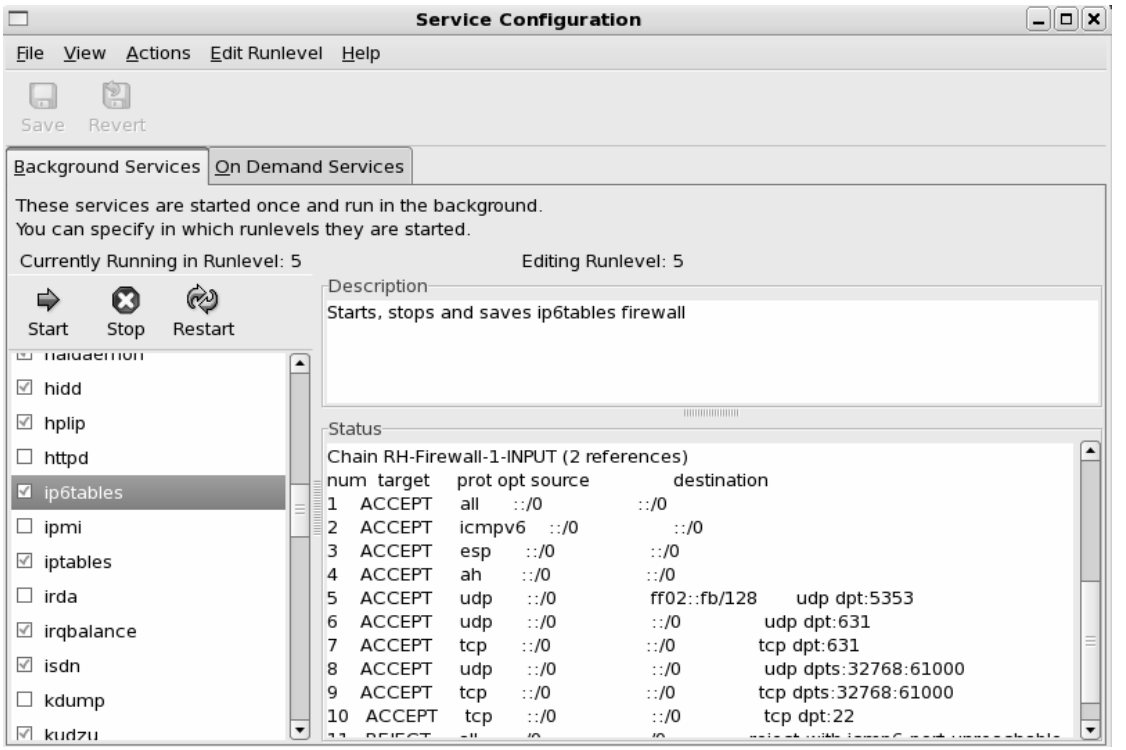
- Role of the server (web, database, proxy, ftp, dns, dhcp or others)
- Services that are required to perform a specific server role (for example, Apache for web server)
- Ports required to be opened (for example, HTTP, port 80)

All the other services should be disabled and all other ports to be closed. When the above tasks are performed, the server becomes a specialized server to play only the designated role.

To harden a server, you must first disable any unnecessary services and ports. This process involves removing any unnecessary services, such as the Linux rlogin service, and locking down unnecessary Transmission Control Protocol/User Datagram Protocol (TCP/UDP) ports. Once these services and ports are secure, you must then regularly maintain the system. Figure 2-4 shows Service Configuration in Red Hat Linux.

System | **Administration** | **Services** opens the **Service Configuration** utility. You may select or deselect the services, start, stop or restart and edit the run level of individual services. In the Figure 2.4 you may notice the service 'ip6tables' is enabled, and the Description of the service and status is displayed.

Figure 2.4 Service Configuration



Though modern Linux distributions have enhanced the GUI to cover most of the administrative tasks, it's essential for good administrators to know how to perform the tasks in the absence of a GUI. Let us discuss about how to manually disable several vulnerable services.

Services to Disable

Linux, by nature, is more secure than most operating systems. Regardless, there are still uncertainties to every new Linux kernel that is released, and many security vulnerabilities that have not been discovered. Most Linux services are not vulnerable to these exploits. However, an administrator can reduce the amount of risk by removing unnecessary services. Red Hat Linux includes many services, so it makes sense that administrators customize the system to suit the company needs. Remember, you are removing risk when you remove unnecessary services.

The xinetd.conf File

Though newer and more sophisticated way managing network services are available in modern Linux distributions, /etc/xinetd.conf file still controls many Unix services, including

File Transfer Protocol (FTP) and Telnet. It determines what services are available to the system. The xinetd (like inetd in earlier versions) service is a "super server"' listening for incoming network activity for a range of services. It determines the actual nature of the service being requested and launches the appropriate server. The primary reason for the design is to avoid having to start and run a large number of low-volume servers. Additionally, xinetd's ability to launch services on demand means that only the needed number of servers is run.

The etc/xinted.conf file directs requests for xinetd services to the /etc/xinetd.d directory. Each xinetd service has a configuration file in the xinetd.d directory. If a service is commented out in its specified configuration file, the service is unavailable. Because xinetd is so powerful, only the root should be able to configure its services.

The /etc/xinetd.d directory makes it simple to disable services that your system is not using. For example, you can disable the FTP and Telnet services by commenting out the FTP and Telnet entries in the respective file and restarting the service. If the service is commented out, it will not restart. The next section demonstrates how to disable the Telnet, FTP, and rlogin services.

Telnet and FTP

Most administrators find it convenient to log in to their Unix machines over a network for administration purposes. This allows the administrator to work remotely while maintaining network services. However, in a high-security environment, only physical access may be permitted for administering a server. In this case, you should disable the Telnet interactive login utility. Once disabled, no one can access the machine via Telnet.

1. To disable Telnet, you must edit the /etc/xinetd.d/telnet file. Open the **Telnet file**, using vi or an editor of your choice.
2. Comment out the **service telnet** line by adding a number sign (**#**) before **service telnet**:

```
#service telnet
```

3. Write and quit the file.
4. Next, you must restart xinetd by entering:

```
/etc/rc.d/init.d/xinetd restart
Stopping xinetd:                          [OK}
Starting xinetd:                          [OK}
```

5. Attempt to log on to the system using Telnet. You should fail.
6. Note that commenting out the service line in the respective xinetd.d directory can disable many services.

7. Disable the FTP service using the same method (e.g., edit the /xinetd.d/wu-ftpd file by commenting out the `service ftp` line and restarting xinetd).
8. Attempt to access the system via FTP. You should be unable to log in to the server.

The Rlogin Service

The remote login (rlogin) service is enabled by default in the /etc/xinetd.d/rlogin file. Rlogin has security vulnerabilities because it can bypass the password prompt to access a system remotely. There are two services associated with rlogin: login and RSH (remote shell). To disable these services, open the **/xinetd.d/rlogin file** and comment out the **service login** line. Then, open the **/etc/xinetd.d/rsh file** and comment out the **service shell** line. Restart xinetd to ensure that your system is no longer offering these services.

Locking Down Ports

TCP/IP networks assign a port to each service, such as HTTP, Simple Mail Transfer Protocol (SMTP), and Post Office Protocol version 3 (POP3). This port is given a number, called a port number, used to link incoming data to the correct service. For example, if a client browser is requesting to view a server's Web page, the request will be directed to port 80 on the server. The Web service receives the request and sends the Web page to the client. Each service is assigned a port number, and each port number has a TCP and UDP port. For example, port 53 is used for the Domain Name System (DNS) and has a TCP port and a UDP port. TCP port 53 is used for zone transfers between DNS servers; UDP port 53 is used for common DNS queries—resolving domain names to IP addresses.

Well-Known and Registered Ports

There are two ranges of ports used for TCP/IP networks: well-known ports and registered ports. The well-known ports are the network services that have been assigned a specific port number (as defined by /etc/services). For example, SMTP is assigned port 25, and HTTP is assigned port 80. Servers listen on the network for requests at the well-known ports. Registered ports are temporary ports, usually used by clients, and will vary each time a service is used. Registered ports are also called ephemeral ports, because they last for only a brief time. The port is then abandoned and can be used by other services.

The port number ranges are classified, as shown in Table 2.1, according to Request for Comments (RFC) 1700. To access RFC 1700, go to ftp://ftp.isi.edu/in-notes/rfc1700.txt. Table 2.2 is a list of well-known TCP/UDP port numbers.

Table 2.1 Port Number Ranges for Various Types

Type	Port Number Range
Well-known	1–1023
Registered	1024–65535

NOTE

Connections to ports number 1023 and below are assumed to run with root-level privileges. This means that untrusted services should never be configured with a port number below 1024.

Table 2.2 Commonly Used Well-Known TCP/UDP Port Numbers

Protocol	Port Number
FTP (Default data)	20
FTP (Connection dialog, control)	21
Telnet	23
SMTP	25
DNS	53
DHCP BOOTP Server	67
DHCP BOOTP Client	68
TFTP	69
Gopher	70
HTTP	80
POP3	110
NNTP	119
NetBIOS Session Service	139
Internet Message Access Protocol (IMAP), version 2	143

Determining Ports to Block

When determining which ports to block on your server, you must first determine which services you require. In most cases, block all ports that are not exclusively required by these services. This is tricky, because you can easily block yourself from services you need, especially services that use ephemeral ports, as explained earlier.

If your server is an exclusive e-mail server running SMTP and IMAP, you can block all TCP ports except ports 25 and 143, respectively. If your server is an exclusive HTTP server, you can block all ports except TCP port 80. In both cases, you can block all UDP ports since SMTP and IMAP all use TCP services exclusively.

However, if you want to use your server as an HTTP client (i.e., for accessing operating system updates) or as an e-mail client to a remote mail server, you will restrict the system by doing this. Clients require registered UDP ports for DNS, as well as registered TCP ports for establishing connections with Web servers.

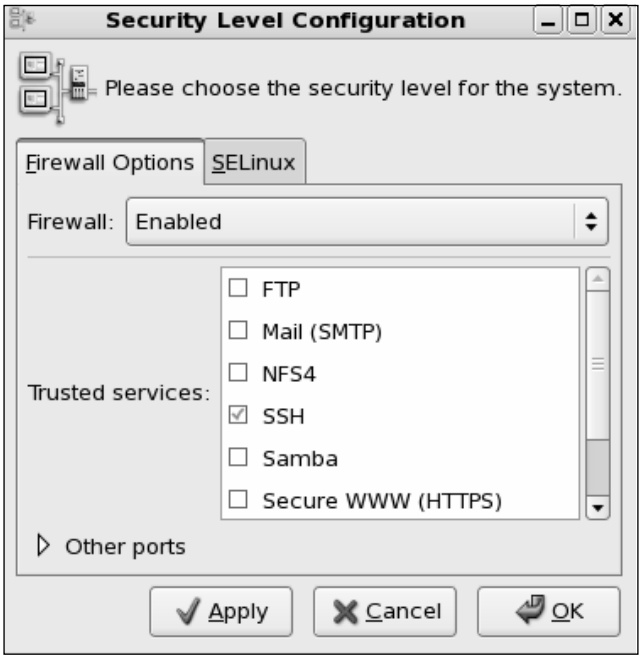
If you open only the corresponding UDP ports 25, 80, and 143, DNS requests are blocked because DNS queries use UDP port 53, and DNS answers use a UDP registered port (e.g., the response stating that www.syngress.com=155.212.56.73). Even if you open port 53, a different registered port may be assigned each time for the answer. Attempting to allow access to a randomly assigned registered port is almost impossible and a waste of time. The same problem applies with TCP connections that require ephemeral ports.

Therefore, you should either open all TCP/UDP registered ports (so you can use your server as a client), or block them (except for the services you require) and access resources, such as operating system updates, another way. You can download the updates from another computer.

Blocking Ports

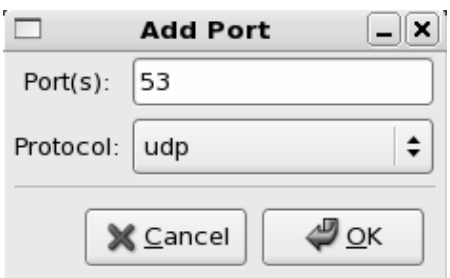
To block TCP/UDP services in Linux, you must disable the service that uses the specific port. You may use the GUI interface of firewall services offered by most of the Linux distributions. In Red Hat Enterprise Linux (RHEL) 5, **System | Administration | Security Level and Firewall** opens up the firewall configuration utility. Figure 2.5 shows the firewall is enabled and the selected services are trusted to run.

Figure 2.5 Security Level & Firewall Configuration



To allow a service to run, just check and enable the service and to block, uncheck the service. If you want to add any non-standard port or a custom port to be allowed by the firewall, then click on **Other ports** and add the protocol type (tcp or udp) and the port number, as shown in Figure 2.6.

Figure 2.6 Adding a Custom Port or Service



The following section discusses disabling ports assigned to stand-alone services.

Stand-Alone Services

To disable ports whose corresponding services are not included in the /etc/xinetd.d directory, you must kill the service's process and make sure that service does not automatically

restart upon reboot. These services are called stand-alone services. For example, port 111 is assigned a stand-alone portmapper service not required for most e-mail servers. The portmapper service, which is technically part of the Sun Remote Procedure Call (RPC) service, runs on server machines and assigns port numbers to RPC packets, such as NIS and NFS packets. Because these RPC services are not used by most e-mail services, port 111 is not necessary. To disable port 111, you must disable the portmapper service as follows:

1. To disable the portmapper service, identify the process identifier (PID) for portmap by entering:

```
ps aux | grep portmap
```

2. The second column lists the PID number. The last column lists the process using that PID. To stop the portmapper service, identify the PID number and enter:

```
kill -9 [PID NUMBER]
```

3. To make sure the service does not restart during reboot, enter:

```
Ntsysv (or use system-config-services gui utility from the terminal window)
```

4. Scroll down to the portmap service and uncheck the check box next to the service. Click **OK**. The portmap service will no longer restart at bootup.

NOTE

Some ports, such as port 80, are not activated unless the service is installed. For example, if you have not installed Apache server, then port 80 is not used. There is no need to block the port because it is already disabled.

Hardening the System with Bastille

Bastille is an open source program that facilitates the hardening of a Linux system. It performs many of the tasks discussed in this chapter such as disabling services and ports that are not required for the system's job functions. The program also offers a wider range of additional services, from installing a firewall (ipchains/iptables) to implementing secure shell (SSH).

Bastille is powerful and can save administrators time from configuring each individual file and program throughout the operating system. Instead, the administrator answers a series of "Yes" and "No" questions through an interactive GUI. The program automatically implements the administrator's preferences based on the answers to the questions.

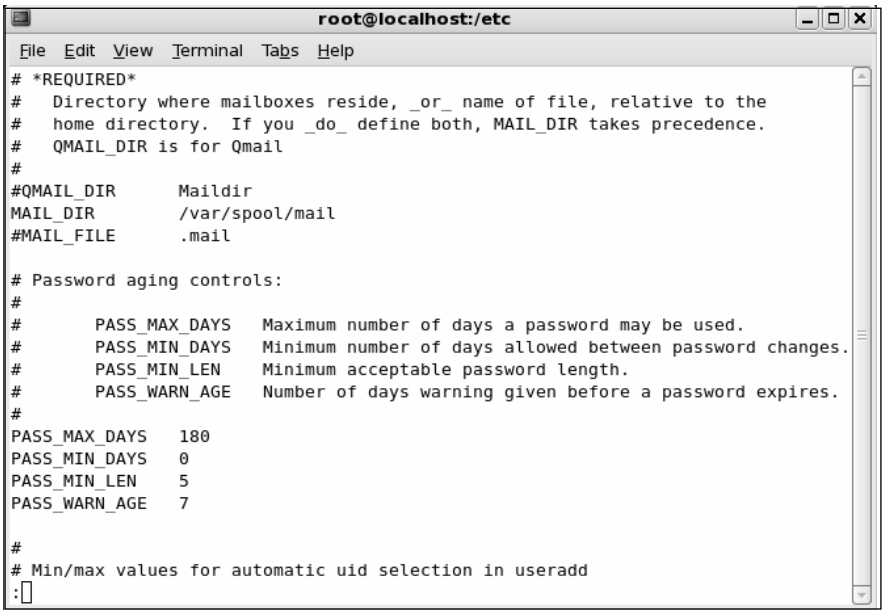
Bastille is written specifically to Red Hat Linux and Mandrake Linux, but can be easily modified to run on most Unix flavors. The specific Red Hat/Mandrake content has been generalized, and now the hard-code filenames are represented as variables. These variables are set automatically at runtime. Before you install Bastille on your system ensure your Linux version is supported by Bastille.

Bastille Functions

The following list highlights the security features offered by Bastille to secure your system. You will choose which feature you want to implement on your system during the question-and-answer wizard. For example, many servers do not need to provide firewall or Network Address Translation (NAT), so you may not need to configure ipchains/iptables. This is a partial list of features offered by Bastille and may vary as new versions of Bastille are released. More information about each of these features is explained in the program.

- **Apply restrictive permissions on administrator utilities** Allows only the root to read and execute common Administrator utilities such as ifconfig, linuxconf, ping, traceroute, and runlevel). It disables the SUID root status for these programs, so nonroot users cannot use them.
- **Disable r-protocols** The r-protocols allow users to log on to remote systems using IP-based authentication. IP-based authentication permits only specific IP addresses to remotely log on to a system. Because this authentication is based on the IP address, a hacker who has discovered an authorized IP address can create *spoofed* packets that appear to be from the authorized system.
- **Implement password aging** Default Red Hat Linux systems allow passwords to expire after 99,999 days. Because this is too long in a secure environment, Bastille offers to change the password expiration time to 180 days. These configurations are written to the /etc/login.defs file, as shown in Figure 2.7.
- **Disable CTRL-ALT-DELETE rebooting** This disallows rebooting the machine by this method.
- **Optimize TCP Wrappers** This choice modifies the inetd.conf (pre-Red Hat Linux 7 versions only) and /etc/hosts.allow files so that inetd must contact TCP Wrappers whenever it gets a request, instead of automatically running the requested service. TCP Wrappers will determine if the requesting IP address is allowed to run the particular service. If the request is not allowed, the request is denied and the attempt is logged. Although IP-based authentication can be vulnerable, this optimization adds a layer of security to the process. This is not recommended for most scenarios.

Figure 2.7 The /etc/login.defs File Configured for 180-Day Password Expiration



- **Add Authorized Use banners** These banners automatically appear whenever anyone logs on to the system. Authorized Use banners are helpful in prosecuting malicious hackers, and should be added to every system on your network that allows access to the network. An information bulletin from the U.S. Department of Energy's Computer Incident Advisory Capability can be found at http://ciac.llnl.gov/ciac/bulletins/j-043.shtml. The bulletin is titled "Creating Login Banners" and explains what is required within login banners for government computers. It also includes how to create banners and provides the text from the approved banner for Federal Government computer systems.
- **Limit system resource usage** If you limit system resource usage, you can reduce the chances of server failure from a DoS attack. If you choose to limit system resource usage in Bastille, the following changes will occur:
  - Individual file size is limited to 40MB.
  - Each individual user is limited to 150 processes.
  - The allowable core files number is configured to zero. Core files are used for system troubleshooting. They are large and exploitable if a hacker gains control of them: they can grow and consume your file system.

- **Restrict console access** Anyone with access to the console has special rights, such as CD-ROM mounting. Bastille can specify which user accounts are allowed to log on via the console.
- **Additional and remote logging** Two additional logs can be added to /var/log/:
  - **/var/log/kernel** (kernel messages)
  - **/var/log/syslog** (error and warning severity messages)

  You can also log to a remote logging host if one exists.
- **Process accounting setup** Allows you to log the commands of all users. It also records when the commands were executed. This log file is helpful in retracing a hacker's steps into your system, but the file can become large quickly. If the hacker has root access, the hacker can remove this accounting log.
- **Deactivate NFS and Samba** Allows you to disable NFS and Samba services. Samba provides a share file system. Unless firewall is configured to block the packets or administrator secures these services Bastille recommends to deactivate these services.
- **Harden Apache Web server** httpd should be deactivated if the service is not required. If you decide to use Apache, you can perform the steps shown in the "Hardening the Apache Web Server" sidebar in Bastille to run the service.

Bastille Versions

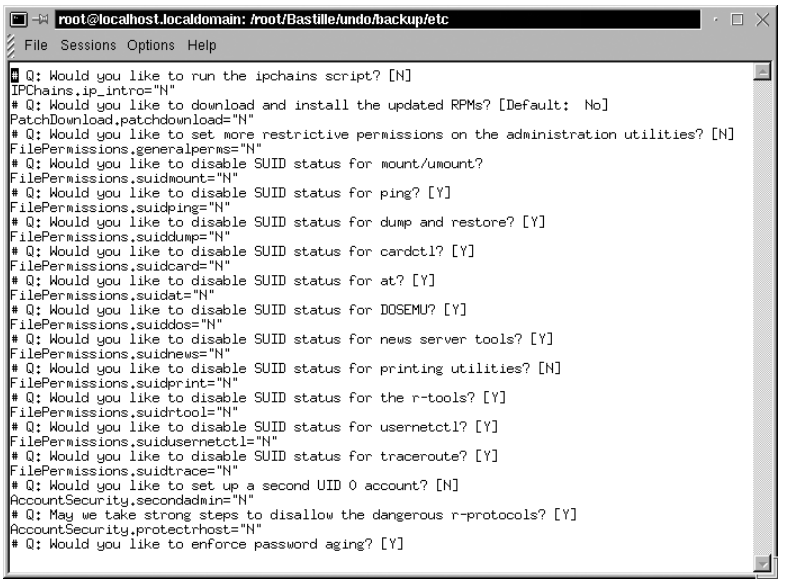
Bastille's current release 3.0.9 incorporates several important changes that make the program even more powerful and easy to use. The examples in this book use Bastille 3.0.9.

Implementing Bastille

Bastille is available for free download at www.sourceforge.net. The program is offered in tarball and rpm format. It must be installed by a root user in his or her root directory (a tarball is a collection of archived files that have been archived using the Unix tar program and have the .tar extension). Ensure perl/tk library is installed on your system as Bastille is a collection of Perl scripts.

The program automatically implements the administrator's preferences based on the answers to the questions, and saves them in the /root/Bastille/config file, as shown in Figure 2.8.

Figure 2.8 Bastille Configuration File



Bastille allows the same configuration to be implemented on other systems. To do this, administrators need to install Bastille on that machine, copy the config file and the BackEnd file to the new system's ~/Bastille directory, and then run the command:

```
#BastilleBackend
```

Damage & Defense…

Logging Your Configurations in Bastille

As with many security programs, Bastille is relatively simple to implement, but it's easy to lose track of the changes you implemented. This can be a problem if you are unable to perform a typical operation on the system, or are denied access to a command or service. Many times, it is because you locked down part of the system by mistake, or misjudged the impact of a particular Bastille choice.

It is always a good idea to create a hard-copy log of the options you select in Bastille, or any security configurations you implement on your system. Create a log with answers given to each question during the implementation and keep the hard copies in a safe place.

If your system goes down, you can access the hard copies and recreate your Bastille configurations. Of course, if your system became unusable due to Bastille, it

Continued

will help you determine what went wrong. This is especially helpful if you are unable to access the /root/Bastille/config file, which saves the administrator's preferences based on the answers to the Bastille questions.

Follow these steps to install and configure Bastille:

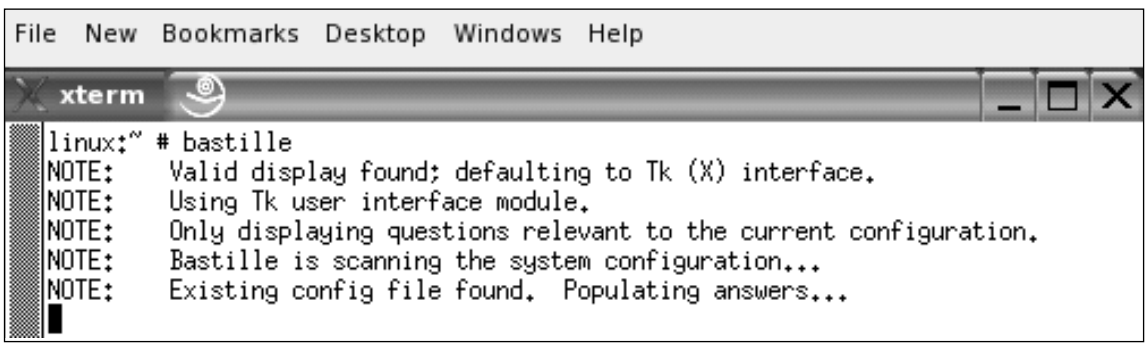
1. Log in as root.
2. Download the rpm file to your root directory. The filename will resemble:

```
Bastille-3.0.9-1.0.noarch.rpm
```

3. Double-click on the package icon (through GUI) or use command line:

```
rpm –i Bastille-3.0.9-1.0.noarch.rpm
```

Figure 2.9 Starting Bastille



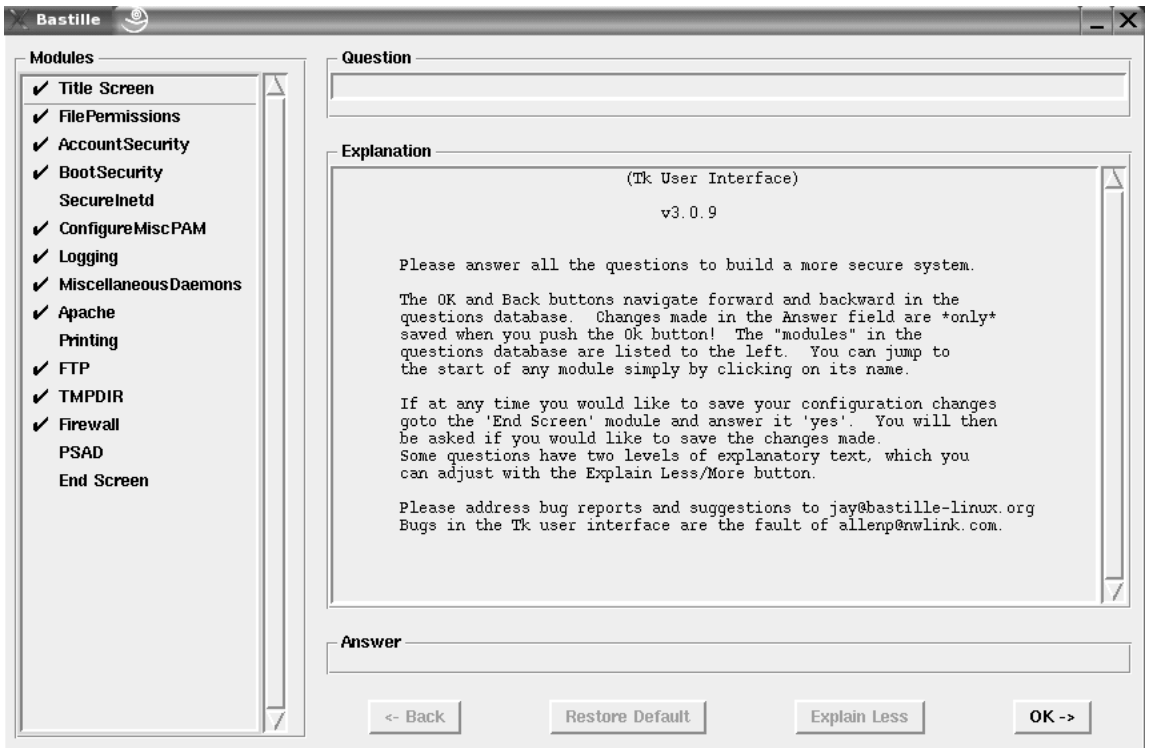
4. To run **Bastille GUI**, enter the following in the Bastille directory (Figure 2.9):

```
./bastille
```

The opening Bastille screen appears, as shown in Figure 2.10.

5. All choices you implement in Bastille are logged to the /root/Bastille/config file. We strongly recommend that you make a backup of the config file before running Bastille and keep a manual log.

Figure 2.10 Bastille GUI



6. The opening screen appears, identifying how to navigate through the Bastille configuration process. Select **Next** to access the first configuration screen, as shown in Figure 2.11.
7. Table 2.5 leads you through the configuration process. You can use Bastille to secure a system based on your system's services and needs. Go through the explanation given below every question and understand the changes Bastille will perform based on your choice.

Figure 2.11 Bastille Linux Question-and-Answer Wizard

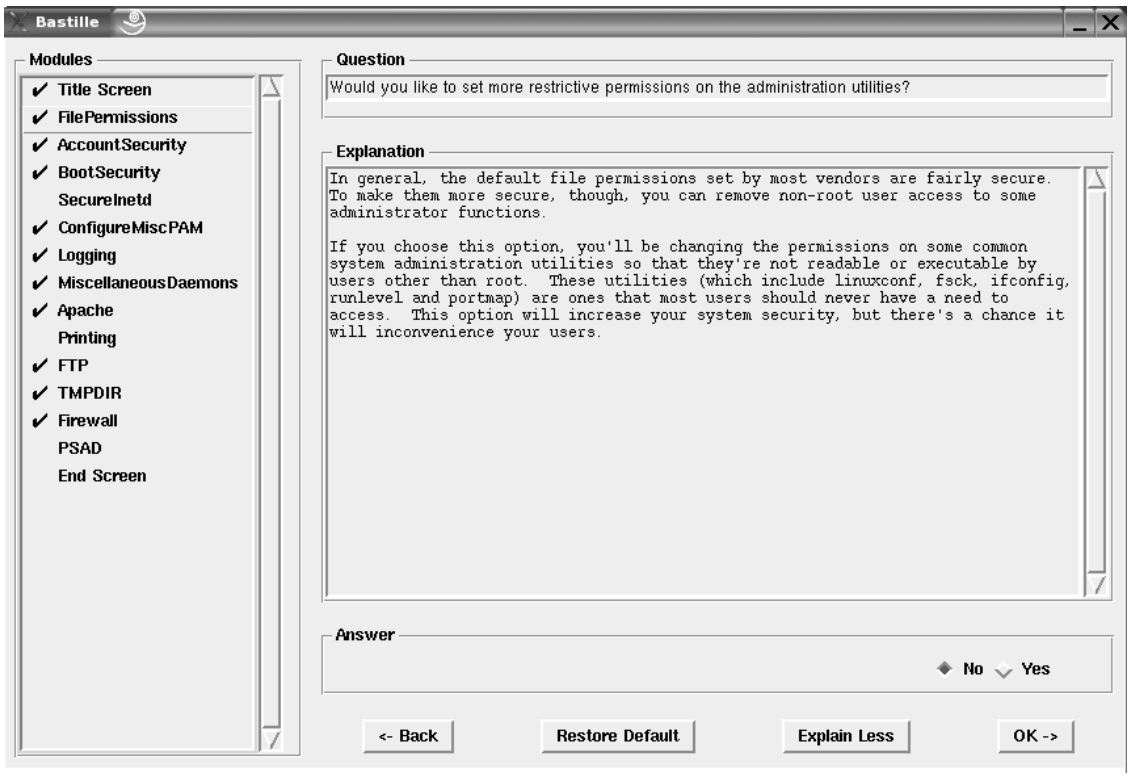


Table 2.5 Bastille Linux Questions

	Questions
1	Would you like to set more restrictive permissions on the administration utilities?
2	Would you like to disable SUID status for mount/umount?
3	Would you like to disable SUID status for ping?
4	Would you like to disable SUID status for at?
5	Would you like to disable the r-tools?
6	Should Bastille disable clear-text r-protocols that use IP-based authentication?
7	Would you like to enforce password aging?
8	Should we disallow root login on tty's 1-6?
9	Would you like to password-protect the GRUB prompt?
10	Would you like to disable CTRL-ALT-DELETE rebooting?

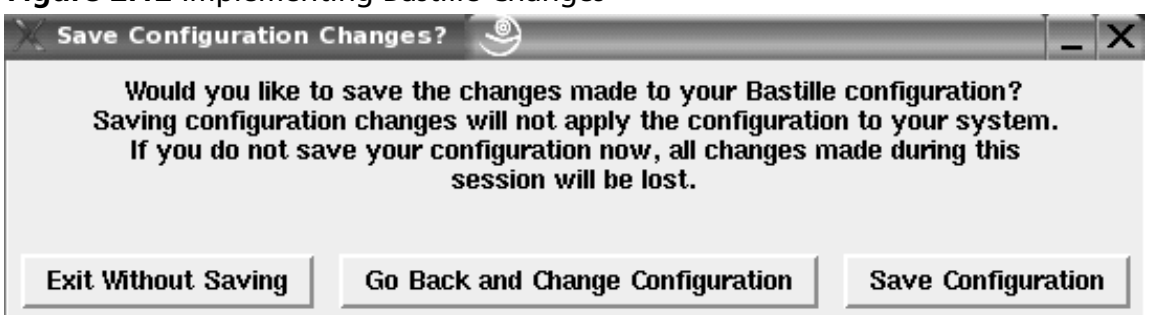
Continued

Table 2.5 continued Bastille Linux Questions

	Questions
11	Would you like to set a default-deny on TCP Wrappers and xinetd?
12	Would you like to display "Authorized Use" messages at log-in time?
13	Who is responsible for granting authorization to use this machine?
14	Would you like to put limits on system resource usage?
15	Should we restrict console access to a small group of use accounts?
16	Would you like to add additional logging?
17	Do you have a remote logging host?
18	Would you like to setup process accounting?
19	Would you like to disable acpid and/or apmd?
20	Would you like to deactivate NFS and Samba?
21	Would you like to deactivate the HP OfficeJet (hpoj) script on this machine?
22	Would you like to deactivate the ISDN script on this machine?
23	Would you like to disable printing?
24	Would you like to install TMPDIR/TMP scripts?
25	Would you like to run the packet filtering script?
26	Are you finished making changes to your Bastille configuration?

8. Bastille asks if you wish to implement these changes, as shown in Figure 2.12.

Figure 2.12 Implementing Bastille Changes



9. Select **Save Configuration** if you want to just save the configuration without applying changes. Select **Exit Without Saving** if you want to discard the changes. Select **Go Back and Change Configuration** if you want to apply the changes.

10. If you implemented password aging to 60 days, observe the changes Bastille made to the login.def file by entering:

```
cat /etc/login.defs | less
```

11. Press any key to display the next page. Press **q** to access the prompt.
12. You applied limits to system resources by limiting individual users to 150 processes, and configuring the allowable core files number to zero. Observe the changes Bastille made to the limits.conf file by entering:

```
cat /etc/security/limits.conf | less
```

13. Press any key to display the next page. Press **q** to access the prompt.

Undoing Bastille Changes

At the time of this writing, a reliable automatic undo feature did not exist in Bastille. To undo the changes, you can run through the configuration questions again and select different answers. There are two other options. There is a Perl script named Undo.pl in the Bastille directory that is designed to undo all changes except for RPM installations. There is also a backup directory located at /root/Bastille/undo/backup that contains all the original system files that Bastille modified. The backup directory structure is the same as the system's directory, so you can manually replace the files fairly easily.

You cannot undo your Bastille configurations by simply removing Bastille. If you do this, your changes will still be written to their specific files. If you want to remove the program and your settings, you must undo your changes, and then remove the Bastille directory.

The following steps demonstrate three ways to undo the changes that you implemented in Table 2.5.

1. One method to undo Bastille configurations is to run through the configuration questions again and select different answers.
2. A second method to undo Bastille configurations is to run the automated Perl script that will undo the changes. The script is named Undo.pl, and is designed to undo all changes except for RPM installations. To run the Undo.pl script, access the **Bastille directory** and enter:

```
./Undo
```

3. A third method to undo Bastille configurations is to manually remove the changes. This can be done by replacing each file that was changed with the backup files in the Bastille directory. The backup directory is located at:

```
/root/Bastille/undo/backup
```

The backup files contain the original files before they were changed, so the original configurations are intact. Bastille makes a backup file of each file before the file is modified.

4. For example, to change password aging back to its default 99,999 days, replace the login.defs file with the backup file. Enter the following:

```
cd /root/Bastille/undo/backup/etc/login.defs
cp logindefs /etc/login.defs
cp: overwrite '/etc/login.defs'? y
```

The backup file replaces the current file, thus returning the password expiration configuration to its default setting.

As you can see, Bastille is a powerful security tool that helps you harden your system. It is relatively simple to use, and can save administrators a great deal of time because it automatically configures the required files for each selection. Administrators do not have to manually write to each file, or disable services individually. Bastille is recommended for any Unix system that offers services, whether it is a LAN or Internet server.

Controlling and Auditing Root Access with Sudo

Superuser Do (sudo) is an open source security tool that allows an administrator to give specific users or groups the ability to run certain commands as root or as another user. Sudo (current release is 1.6.9p5) is available for download from www.gratisoft.us/sudo/download.html. The program can also log commands and arguments entered by specified system users. The developers of sudo state that the basic philosophy of the program is to "give as few privileges as possible but still allow people to get their work done." Sudo was first released to the public in the summer of 1986. The program is distributed freely under an ISC-style license. The Sudo Main Page is located at http://www.gratisoft.us/sudo/, as shown in Figure 2.13.

The program is a command-line tool that operates one command at a time. Table 2.6 lists several important features of sudo.

Figure 2.13 Sudo Home Page

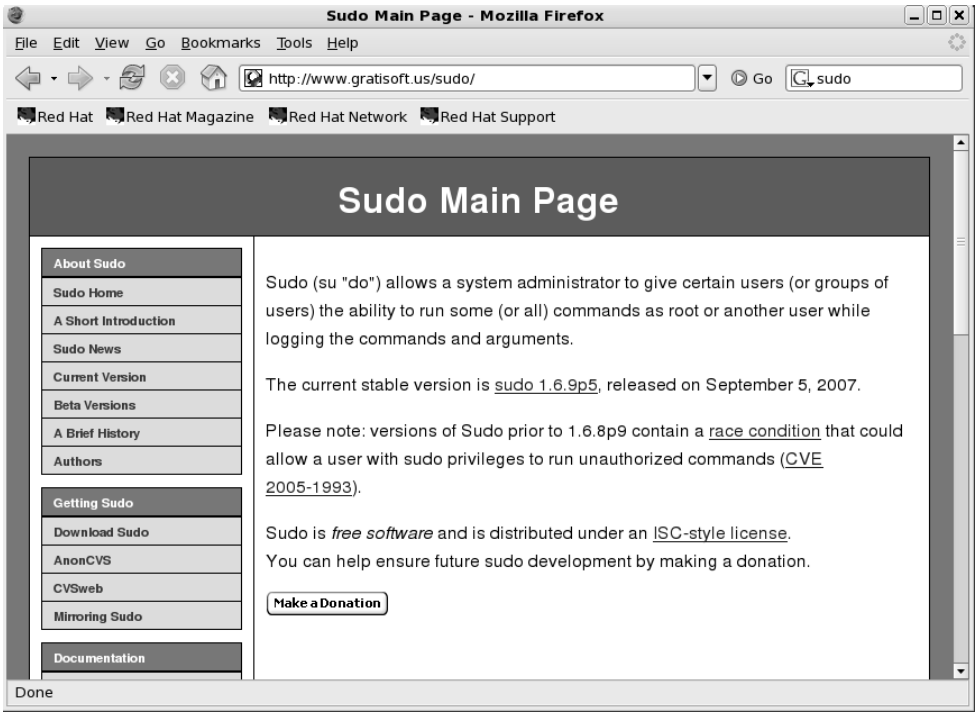


Table 2.6 Sudo Features

Feature	Description
Command logging	Commands and argument can be logged. Commands entered can be traced to the user. Ideal for system auditing.
Centralized logging of multiple systems	Sudo can be used with the system log daemon (syslog) to log all commands to a central host.
Command restrictions	Each user or group of users can be limited to what commands they are allowed to enter on the system.
Ticketing system	The ticketing system sets a time limit by creating a ticket when a user logs on to sudo. The ticket is valid for a configurable amount of time. Each new command refreshes the ticket for the predefined amount of time. The default time is five minutes.
Centralized administrationof multiple systems	The sudo configurations are written to the /etc/sudoers file. This file can be used on multiple systems and allows administration from a central host. The file is designed to allow user privileges on a host-by-host basis.

Because sudo logs all commands run as root (or specified otherwise), many administrators use it instead of using the root shell. This allows them to log their own commands for troubleshooting and additional security.

The ticketing system is ideal because if the root user walks away from the system while still logged in (a very bad idea), another user cannot access the system simply because he or she has physical access to the keyboard.

After the ticket expires, users must log on to the system again. A shorter time is recommended, such as the default five minutes. The ticketing system also allows users to remove their ticket file.

System Requirements

To install and run sudo from the source distribution, you must have a system running Unix. Almost all versions of Unix support the sudo source distribution, including almost all flavors of POSIX, BSD, and SYSV. You must also install the C compiler and the make utility.

Sudo is known to run on the following Unix flavors: Auspex, SunOS, Solaris, ISC, RISCos, SCO, HP-UX, Ultrix, IRIX, NEXTSTEP, DEC Unix, AIX, ConvexOS, BSD/OS, OpenBSD, Linux, UnixWare, Pyramid, ATT, SINIX, ReliantUNIX, NCR, Unicos, DG/UX, Dynix/ptx, DC-Osx, HI-UX/MPP, SVR4, and NonStop-UX. It also runs on MacOSX Server.

The Sudo Command

The **sudo** command allows a user to execute a command as a superuser or another user. All configurations for sudo are written to the /etc/sudoers file. The sudoers file specifies whether that command is allowed by that particular user.

In order to use sudo, the user must have already supplied a username and password. If a user attempts to run the command via sudo and that user is not in the sudoers file, an e-mail is automatically sent to the administrator, indicating that an unauthorized user is accessing the system.

Once a user logs in to sudo, a ticket is issued that is valid by default for five minutes. A user can update the ticket by issuing the **–v** flag, which will validate the ticket for another five minutes. The command is entered as follows:

```
sudo -v
```

If an unauthorized user runs the **–v** flag, an e-mail will not be sent to the administrator. The **–v** flag informs the unauthorized user that he or she is not a valid user. If the user enters command via sudo anyway, an e-mail will then be sent to the administrator.

Sudo logs login attempts, successful and unsuccessful, to the syslog(3) file by default. However, this can be changed during sudo configuration. Some of the command-line options listed in Table 2.7 are used by sudo.

Table 2.7 Selected Sudo Command Options

Option	Option Name	Description
-V	Version	Prints version number and exits.
-l	List	Lists the commands that are allowed and denied by current user.
-h	Help	Prints usage message and exits.
-v	Validate	Updates the user's ticket for a configured amount of time (default is five minutes). If required, the user must re-enter the user password.
-k	Kill	Expires the user's ticket. Completing this option requires the user to re-enter the user password to update the ticket.
-K	Sure kill	Removes the user's ticket entirely. User must log in with username and password after running this option.
-u	User	Runs the specific command as the username specified. The user specified can be any user except root. If you want to enter a uid, enter **#uid** instead of the username.

Installing Sudo

Download Sudor tarball from www.gratisoft.us/sudo/download.html to any directory you choose. Sudo has been downloaded to the /root directory for this example. This exercise was performed on a Red Hat Enterprise Linux version 5.0.

1. Access the directory where you downloaded sudo, and decompress the tar file (your sudo version number will vary depending on the version of sudo that you downloaded) by entering:

```
tar -zxvf sudo-1.6.3p5.tar.gz
```

2. A directory will be created, such as sudo-1.6.3p5.
3. Access the sudo directory by entering:

```
cd sudo-1.6.3p5
```

4. To create a makefile and config.h file that will allow you to configure sudo, enter:

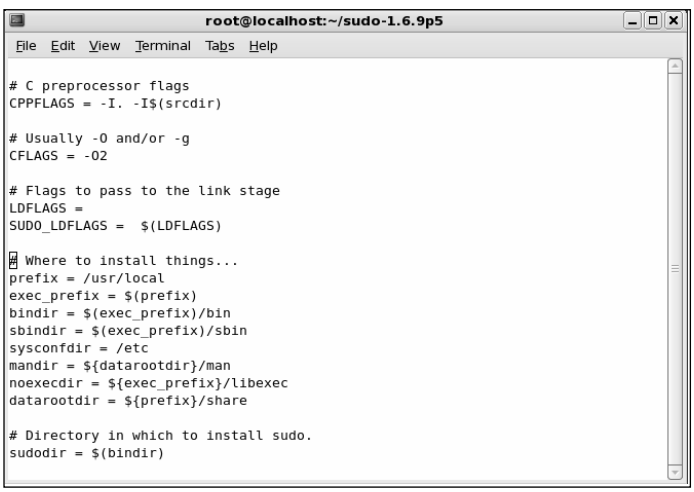
```
./configure
```

5. You can add options to the **./configure** command to customize your sudo installation. Simply append the options to your **./configure** command. The entire list of options is available in the /sudo/INSTALL file.
6. You can also edit makefile to change the default paths for installation, as well as the other configurations listed in /sudo/INSTALL file. If you require this change, open **makefile** in a text editor. For example, enter:

```
vi Makefile
```

7. Locate the "Where to install things..." section of makefile, as shown in Figure 2.14.

Figure 2.14 Sudo Makefile



8. Change the default paths if necessary. For this example, we recommend that you use the default paths.
9. Quit the file. If you use the vi text editor, enter:

```
:q
```

10. (Optional) You can also change the default installation paths when you run the **./configure** command (you ran the **configure** command in a previous step). To do this, enter an option after the command. For example, by default the sudoers file is installed in the /etc directory. You can change this location by entering:

```
./configure - -sysconfdir=DIR
```

where DIR is the new installation directory.

11. To compile sudo, run the **make** command by entering:

```
make
```

12. (Optional) You will probably need GNU if you install sudo in a directory other than the source file directory. If you have errors during installation, read the TROUBLESHOOTING and PORTING files.
13. To install sudo, you must be the root user. Run the **make install** command to install the man pages, visudo, and a basic sudoers file by entering:

```
make install
```

NOTE

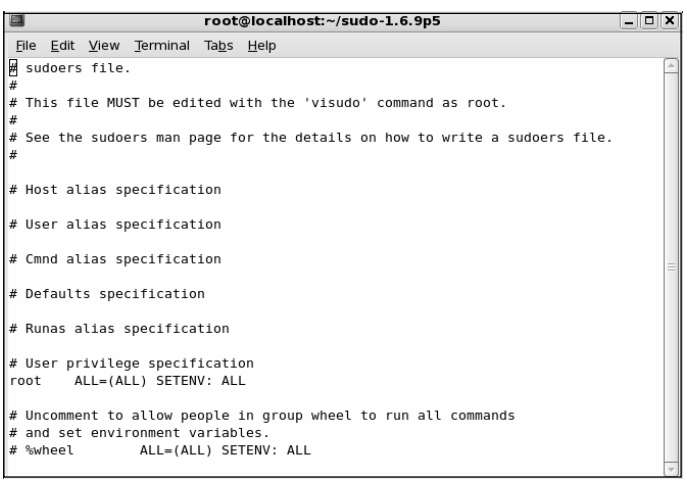
Any existing sudoers file will not be overwritten.

14. You have installed sudo. The next section explains how to configure it to suit your system's needs.

Configuring Sudo

To configure sudo, you must edit the %/sudo-1.6.9p5/sudoers file. The sudoers file defines which users are allowed to execute what commands. Only the root user is allowed to edit the file, and it must be edited with the **visudo** command. A sample.sudoers file is included in the sudo directory, and is shown in Figure 2.15.

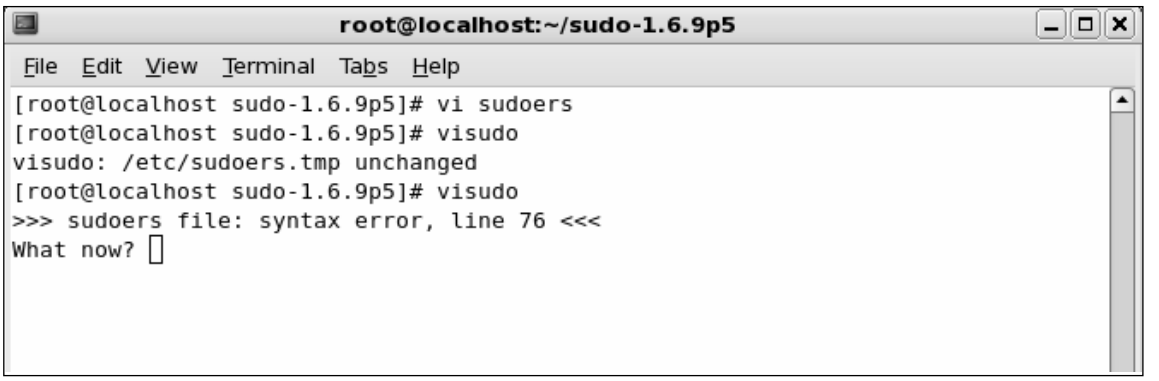
Figure 2.15 Sample.Sudoers File



The **visudo** command opens the sudoers file, by default, in the vi text editor. The **vi** commands are used to edit and write the file. You can change the default text editor used by visudo using the compile time option. Visudo uses the EDITOR environment variable. The **visudo** command performs the following tasks when editing the sudoers file:

- **Checks for parse errors** Visudo will not save any changes if a syntax error exists. It will state the line number of the error and prompt you for guidance. You will be offered a "What Now?" prompt and three choices: "e" to re-edit the file, "x" to exit without saving, and "Q" to quit and save changes. A syntax error result is shown in Figure 2.16.

Figure 2.16 Visudo Parse Error



NOTE

If a syntax error exists in the sudoers file and you choose **Q** to quit and save the visudo changes, sudo will not run until the problem is corrected. You must run visudo again, fix the problem, and save the file again. It is recommended that you select **e** to attempt to fix the problem, or **x** to exit without saving (if you are not sure of what went wrong).

- **Prevents multiple edits to the file simultaneously** If you attempt to run visudo while the sudoers file is being edited, you will receive an error message informing you to try again at a later time

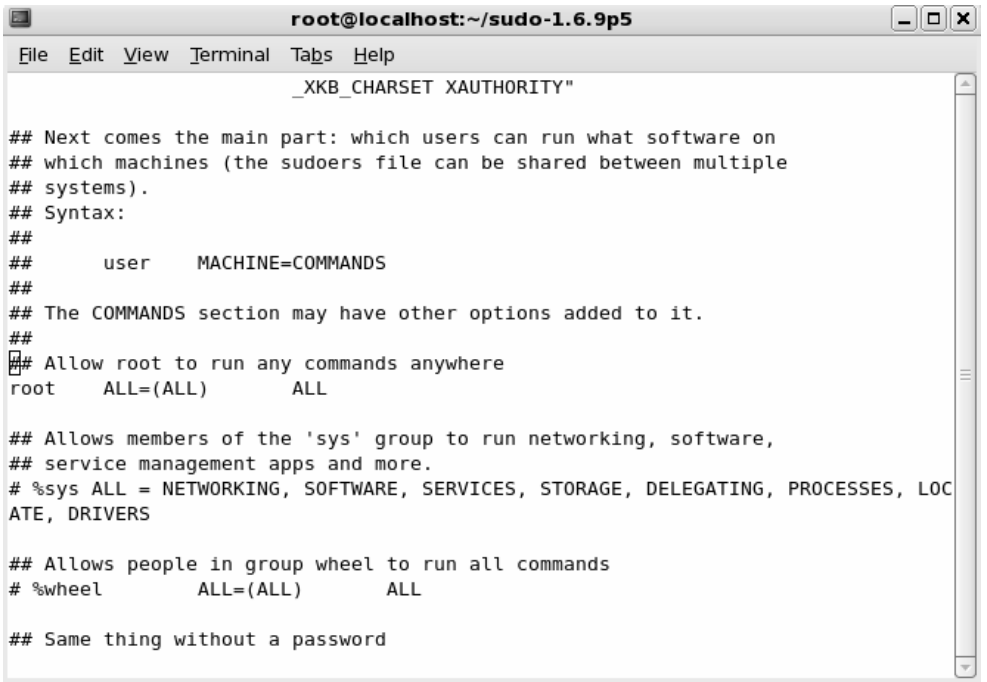
The sudoers file consists of two different types of entries, *user specifications* and *aliases*. The following examples show you how to use user specifications, which define which user is allowed to run what commands. Aliases are basically variables.

The sudoers file contains a root entry. The default sudoers file is shown in Figure 2.17. The user privilege specification is listed as

```
root    ALL=(ALL) ALL
```

This configuration allows the root user to issue all commands.

Figure 2.17 Default Sudoers File Allowing the Root User Access to All Commands



To allow other users to run commands as root, you must enter those users in the sudoers file. You must also list the host on which they are allowed to run the commands. Last, you must list the specific commands that those users are allowed to run as root. In the following steps, you will create user *bob* and allow him to run several commands as root using sudo on your system.

1. Open the sudoers file by entering:

```
visudo
```

2. The sudoers file opens in vi. Locate the "User privilege specification" section. After the root entry, enter the following (press **i** to insert text):

```
bob    your-hostname = /sbin/ifconfig, /bin/kill, /bin/ls
```

3. This line allows user bob to run the **ifconfig**, **kill** and **ls** commands as root.

NOTE

By default, all commands you list in sudoers will run as root unless you specify otherwise. For example, bob could run commands as user *bugman* if desired. You would enter:

```
bob your-hostname = (bugman) /sbin/ifconfig
```

In this case, the **ifconfig** command will run as user bugman. You can allow bob to enter commands as several different users.

```
bob your-hostname = (bugman) /sbin/ifconfig, (root) /bin/kill, /bin/ls
```

The **kill** and **ls** commands will run as root, while the **ifconfig** command runs as bugman. At the command line, bob will enter:

```
sudo –u bugman /sbin/ifconfig
```

3. Press **ESC** to write and quit the file. Then, enter:

```
:wq
```

This command writes and quits the file using vi.

4. Now you must create user bob. Enter:

```
useradd bob
```

5. Create a password for user bob by entering:

```
passwd bob
Changing password for user bob
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

Running Sudo

You have configured sudo to allow user bob root privileges for the **ifconfig**, **kill**, and **ls** commands. When bob wants to run these commands, he must first enter the **sudo** command, and then his password.

1. Log on as user bob.
2. To find out what commands bob has root access to, enter the following:

```
sudo –l
```

3. If this is your first time running sudo as user bob, a warning will display:

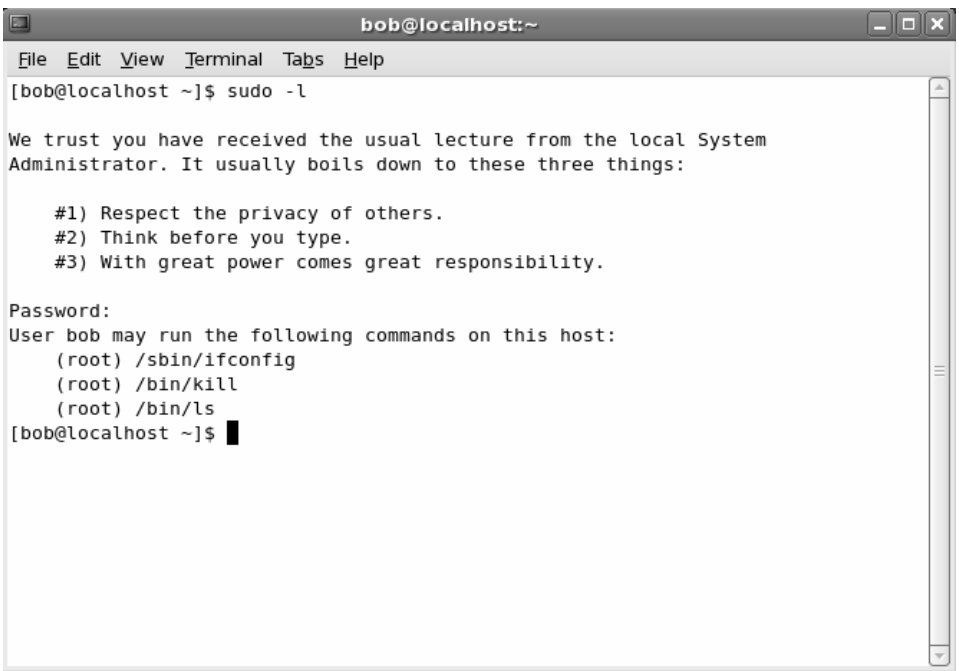
```
We trust you have received the usual lecture from the local System Administrator.
It usually boils down to these three things:
#1) Respect the privacy of others.
#2) Think before you type
#3) With great power comes great responsibility
```

4. A password prompt appears. Do *not* enter the root password. Enter bob's password.

```
Password:
```

5. The commands that bob is allowed to run on this host are listed, as shown in Figure 2.18.

Figure 2.18 Commands That User Bob Can Run as Root



6. To test your sudo configurations, run an ifconfig option that requires root permission without using sudo. Enter:

```
/sbin/ifconfig eth0 down
```

Permission is denied because bob is not allowed to deactivate the system's interface.

7. To deactivate the interface, bob must use sudo. Enter:

```
sudo /sbin/ifconfig eth0 down
```

You will be successful. Please note that sudo will ask for the bob's password if bob's ticket has expired (the default is five minutes). If you run this command within five minutes from the last, you will not be prompted for a password.

8. Reactivate the interface. Enter:

```
sudo /sbin/ifconfig eth0 up
```

9. Next, restart one of the httpd processes using the **kill** command by entering:

```
ps aux | grep httpd
```

10. Choose an Apache PID from the list that appears (If Apache is not installed, select a different service process to restart). Enter:

```
kill –HUP [PID NUMBER]
```

11. You are not allowed to restart the httpd process because you are not root. You will receive the following result:

```
bash: kill: (PID NUMBER) - Not owner
```

12. Instead, use sudo to run the command as root by entering:

```
sudo kill –HUP (PID NUMBER)
```

You should be successful.

13. Next, you will list the root user directory as user bob using the **ls** command. Enter:

```
 ls /root
```

Permission is denied because you are not root.

14. Again, use sudo to run the command as root:

```
sudo ls /root
```

Permission is granted and the root user's directory is displayed.

15. To expire bob's timestamp, enter the command **sudo –k**. Bob will have to enter a password the next time he uses sudo.

No Password

In some situations, entering a password each time sudo is run is redundant because the user has already logged on to the system. Sudo offers a way around this monotonous task by using the NOPASSWD tag in the sudoers file.

1. To remove the password requirement in the sudoers file, log on as *root* and enter:

```
visudo
```

2. The sudoers file opens in vi. Modify bob's user privilege specification to match the following (press **i** to insert text):

```
bob   your-hostname = NOPASSWD: /sbin/ifconfig, /bin/kill, /bin/ls
```

3. Press **ESC**. Enter **:wq** to write and quit the file.
4. Log on as bob. Deactivate the interface using sudo:

```
sudo /sbin/ifconfig eth0 down
```

You will not be prompted for your password and the command will run as root.

5. Reactivate the interface. Enter:

```
sudo /sbin/ifconfig eth0 up
```

Sudo Logging

As mentioned previously, sudo logs which users run what commands. Logging does not occur automatically. You must set up sudo and syslogd to log commands. This involves two steps. First, you must create a sudo logfile in /var/log. Second, you must configure syslog.conf to log sudo commands. The following steps show you how to configure sudo logging.

1. Log on as root. Create a sudo log file in /var/log/. Enter:

```
touch /var/log/sudo
```

2. Next, you must add a line in the syslog.conf file to direct logging to your sudo logging file. Open syslog.conf by entering the following:

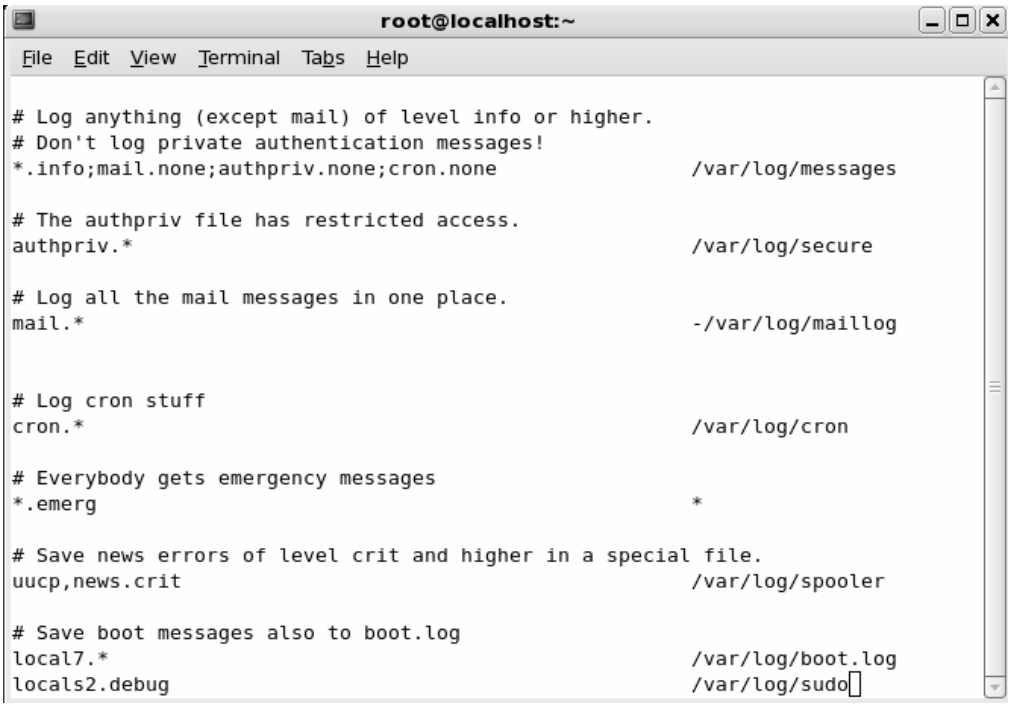
```
vi /etc/syslog.conf
```

3. Enter the following line at the end of the syslog.conf file (press **i** to insert text). The white space must be created using TAB, not the SPACE BAR.

```
local2.debug                                /var/log/sudo
```

4. This syslog.conf entry logs all successful and unsuccessful sudo commands to the /var/log/sudo file. You can also log to a network host by indicating the network host instead of a local directory. The syslog.conf file is shown in Figure 2.19.

Figure 2.19 Editing the Syslog.conf file for Sudo Logging



5. Press **ESC** to write and quit the file. Then, enter:

```
:wq
```

6. Since you have modified the syslog.conf file, you need to restart syslogd. To send a HUP signal to syslogd, you must first know the syslogd process identifier (PID). To identify the syslogd PID, enter:

```
ps aux | grep syslogd
```

7. The second column lists the PID number. The last column lists the process using that PID. To restart syslogd, identify the PID number and enter:

```
kill -HUP [PID NUMBER]
```

8. First, you will generate log entries for user bob. Log on as user bob.
9. Enter the following **ifconfig** commands while logged on as user bob:

```
sudo -l
sudo /sbin/ifconfig eth0 down
sudo /sbin/ifconfig eth0 up
```

10. Restart one of the httpd processes (or another process) using the **kill** command by entering:

```
ps aux | grep httpd
```

11. Choose an Apache (httpd) PID from the list that appears. Enter:

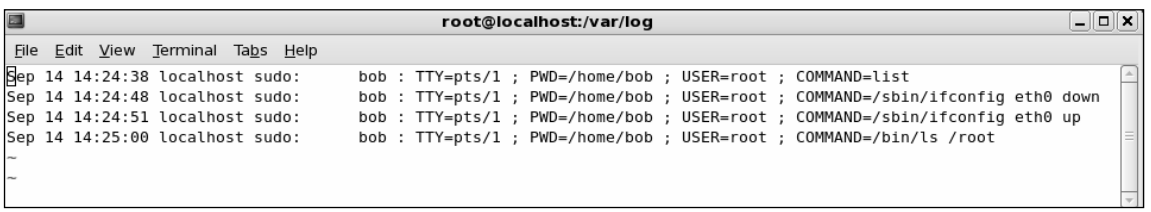
```
sudo kill –HUP [PID NUMBER]
```

12. Now list the root user directory as user bob. Enter:

```
sudo ls /root
```

13. Log on as root and view the sudo log file. All the sudo commands that bob entered are listed, as shown in Figure 2.20.

Figure 2.20 Sudo Log File Displaying User Bob's Commands



14. You can log any root commands by simply typing **sudo** before each command. For example, make sure that you are logged on as root and enter the following commands (or any commands you choose):

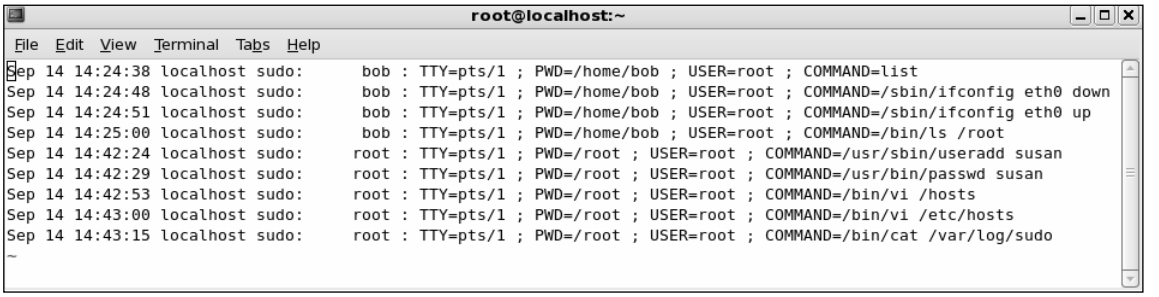
```
sudo useradd susan
sudo passwd susan
sudo vi /hosts
```

15. Access and view the sudo log file by entering:

```
sudo cat /var/log/sudo
```

All root user entries are logged, including the **cat** command you just entered, as shown in Figure 2.21.

Figure 2.21 Sudo Log File Displaying Root User Commands



As you can see, sudo is extremely helpful for controlling and auditing root access. It allows a system administrator to distribute root system tasks without distributing the root password. An administrator can control what root access is needed for each user, and can customize system access based on those needs.

Sudo is used almost entirely by system administrators, and is a great way to train new system administrators. New administrators can be given a new account with only selected root privileges. The master administrator can then review the work of the administrator in training.

This section discussed one of the many ways to use sudo; it focused primarily on user specifications in the sudoers file. For more information on extending sudo, such as using aliases, please consult the sudo man file available at www.gratisoft.us/sudo/man/sudo.html.

Managing Your Log Files

Another aspect of system security is managing your log files. By default, Linux offer modest logging so that administrators can see who and what has accessed their system. More logging is available (both more detail and logging on more services), but Linux keeps it brief so that you don't fill your hard disk with log information. This section briefly discusses helpful commands and programs that provide access to system logs.

Linux offers commands that allow administrators to access useful log files. Two commands of interest are **last** and **lastlog**. The message file also offers useful data for determining possible security breaches on your system.

The **last** command displays data such as who is logged on to the system, who recently logged on, and when the system has rebooted. For example, you may receive data such as the following:

```
root   pts/1  :0.0   Tue Sep 18 01:13     still logged in

root   pts/1  :0.0   Tue Sep 18 01:02 - 01:06    (00:03)

root   :0            Tue Sep 18 01:00     still logged in
```

The **lastlog** command displays the users and services that have accounts on your machine. It lists the last time each account logged in to the system, or if the account has ever logged in. Each service in Linux is given an account. This is very helpful because if a service logged in without your knowledge, a hacker may be responsible. This would indicate that the hacker controls your system and is currently exploiting it. It could also mean that another administrator started the service without telling you.

The messages file is a log file that displays a list of recent activity on the system. For example, it lists if a password was changed and who changed it. It identifies when a user session opens and closes. It also lists the time and data each event took place. It can be viewed by entering the following command:

```
tail /var/log/messages
```

If you prefer a GUI to view your log files, a program called *SWATCH* (not installed by default) allows an instant and real-time display for various log files. It can view any log files you specify and is discussed in the next section.

The Linux logs should be checked frequently to determine if any security violations have occurred on your system. Logs do not offer solutions, so you must analyze the data and decide how to counteract the attack.

Using Logging Enhancers

Logging enhancers are tools that simplify logging by allowing logging information to be filtered and often displaying logs in simplified formats. Many open source logging programs exist to make system administration much easier. Viewing text-based files with hundreds or thousands of entries can be burdensome, especially if you are only looking for one specific error entry. Logging enhancers can make logging a much more user-friendly experience, and greatly expand and customize the information you need to log.

The next sections explain three popular logging services used by administrators: SWATCH, scanlogd, and the next generation of syslogd (syslogd-ng).

SWATCH

Simple WATCHer or Simple WATCHdog (SWATCH) is an open source package that allows administrators to efficiently monitor system activity. It can monitor events on a system, or a large number of systems, by monitoring system logs for specified events. SWATCH'S main function is to monitor messages actively as they are written to a log files through the Unix syslog utility. SWATCH requires Perl 5 to function.

SWATCH is efficient because it allows administrators to modify the SWATCH configuration file (/etc/swatchrc) to filter logging entries and respond to certain events. For example, SWATCH can monitor the system for bad login attempts, and e-mail the administrator whenever this failed authentication event occurs. It can monitor and alter when

system halts and reboots occur, when a user upgrades to root using the **su** command, when the file system is full, and when someone is sniffing the system. It can monitor anything desired from the log files.

To learn about SWATCH and download the program, you need to visit the SWATCH home page at http://swatch.sourceforge.net . The SWATCH home page is shown in Figure 2.22.

Figure 2.22 SWATCH Home Page



NOTE

At the time of this writing, version 3.2.2 of SWATCH was available for download.

SWATCH uses two required fields: *pattern(s)* and *action(s)*.

- **Patterns** The SWATCH configuration file looks for *patterns* in logging entries. For example, bad login attempts display a "Failed Authentication" error.

- **Actions** Whenever a pattern is discovered, SWATCH seeks an *action*, such as e-mailing the administrator of the failed login attempt.

Two optional fields are used to further customize the configuration file:

- **Throttle** Throttle determines the amount of time that SWATCH will ignore repeated logged entries before listing the entry again. This saves administrators' time from viewing 300 identical "Failed Authentication" errors. However, a secure system should limit the number of login attempts. The throttle entry is defined as HH:MM:SS, where H represents hours, M represents minutes, and S represents seconds.
- **Timestamp** Timestamp defines the length and location of the timestamp. The timestamp entry is defined as start:length.

The following are two examples from the SWATCH configuration file. SWATCH will actively watch for these messages as they are written to their respective log files through the syslog utility. The first example monitors logging for failed login attempts and e-mails root when a failed login attempt occurs.

```
#Failed login attempts
watchfor      /failed/
              echo bold
              mail addresses=root,subject=Failed Authentication
```

The second example monitors your log files and e-mails root when a user sued to gain root access.

```
#Users sued to gain root access
watchfor      /su:/
              echo bold
              mail addresses=root,subject=User sued to root
```

SWATCH filters logging files so that administrators only receive the information they require. It saves a lot of time and trouble once configured and is recommended for system administrators who are overwhelmed by log files (and perhaps do not use them for that reason). To download an RPM version of SWATCH, visit www.rpmfind.net.

Scanlogd

Scanlogd is an open source program that detects and logs TCP-port scanning on a system. For example, it can detect nmap scans. Nmap is a program used by hackers to create a "map" of your network. It is often the first step a hacker takes once he or she has access to your network to determine which system to hack. Nmap lists the systems and the services

on the network. Scanlogd can alert an administrator when the network is being mapped, but it cannot stop the intrusion.

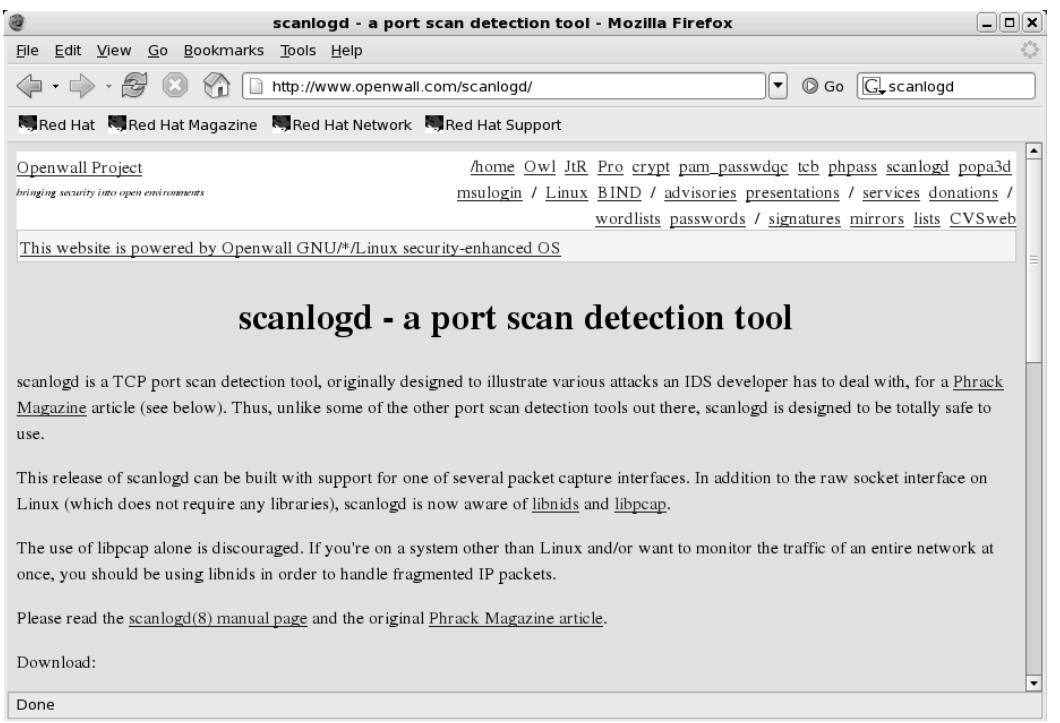


SECURITY ALERT!

Scanlogd was originally designed to illustrate attacks, not to fix them. Therefore, even though it is safe to run on your system, it does not prevent hacking attacks. You must read the system log to discover what happened to your system, and then determine the appropriate solution.

Scanlogd writes one line per scan using the syslog(3) mechanism. It also logs when a source address sends many packets to several different ports in a short amount of time. You can learn about scanlogd and download the program at www.openwall.com/scanlogd. The scanlogd home page is shown in Figure 2.23.

Figure 2.23 Scanlogd Home Page



Because scanlogd is only meant to detect scans, it is totally safe to run on your system. It must have access to raw IP packets to function, and can capture packets coming in and out

of the system interface, or across the network to which the system is attached. In addition, scanlogd supports the raw socket interface on libnids, libpcap, and Linux.

Syslogd-ng

Syslogd-ng is a logging daemon that is the replacement for the traditional syslogd. The "ng" is an acronym for "next generation." The original syslogd was the general Unix logging daemon that handled requests for syslog services, but was difficult to configure. Syslogd-ng is easier to configure and offers additional logging features, such as more configurations. For example, syslogd-ng allows administrators to filter messages based on priority, as well as the content of the messages. You can also forward logs on TCP, sort logs to different destinations, and create a direct log stream to various hosts. The syslog-ng home page is shown in Figure 2.245 and is located at www.balabit.com/network-security/syslog-ng/.

Figure 2.24 Syslog-ng Home Page

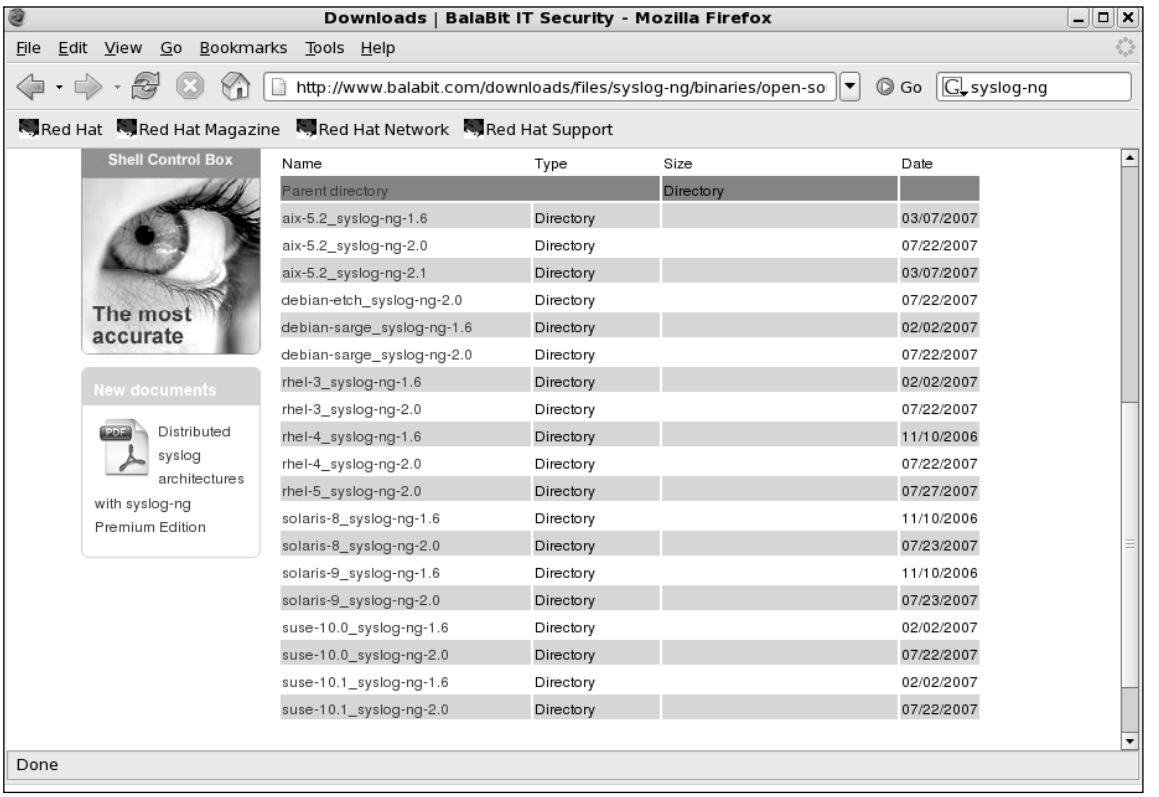


The basic problem with system logs is that they contain a lot of unimportant information. This information is often called *noise*. Many events are lost because they are buried in the noise. Syslogd made it difficult to choose only the important messages.

The reason this occurs is that messages are sent to different destinations depending on the assigned facility/priority pair. These destinations are very broad, and include general

facilities such as mail, news, auth, and so forth, and priorities ranging from alert to debug. Many programs use the facilities; so many unneeded messages are written to their logs. In many cases, the message and the facility are not even related. Syslogd-ng filters messages based on message content in addition to the facility/priority pair. Using this method, only the messages that are needed are logged.

Figure 2.25 Syslog-ng Platforms



Syslogd-ng is available on AIX, Debian, Red Hat Enterprise Linux 3, 4, 5, Sun Solaris 8, 9, and Suse Linux 10. At the time of this writing, the latest stable version was 2.0. You can learn more about syslog-ng and download manuals from www.balabit.com/support/documentation/. The product is available in Open Source and Premium editions. You may also request for an evaluation before deploying it in your production network. Additional support packages (such as libdi8) and database specific binaries may be required. Figure 2-26 shows supported versions available for download from Balabit.

Security Enhanced Linux

Security Enhanced Linux or SELinux is a research project initiative from NSA (National Security Agency). Recommendations of this project are incorporated in various Linux distributions. The project focuses on utilizing mandatory access control (MAC) architecture incorporated into the kernel subsystems rather than working on the security vulnerabilities on the operating system itself. More information on SELinux, whitepapers, presentation and downloads are available at www.nsa.gov/selinux/index.cfm.

Red Hat Linux has incorporated SELinux in its releases. If you have not enabled SELinux during the installation stage, the same can be enabled from

Let's look at the components of SELinux specific to Red Hat Enterprise Linux (RHEL) in version 5. Before that you need to enable SELinux.

1. To enable SELinux click on **System** | **Administration** | **Security Level and Firewall**
2. Click on the second tab **SELinux**. You will find three options Enforcing (policy enforced state), Permissive (warnings only, no policy enforcement) and Disabled (SELinux policy fully disabled state). Once you have enabled SELinux, the system may re-label the file system for SELinux. This requires a restart.
3. To manage SELinux click on **System** | **Administration** | **SELinux Management Tool**. This tool has status, boolean, file labeling, user mapping, SELinux user, translation, network port and policy module configuration options. First screen that opens is the status screen. This includes the current status for system default enforcing mode, current enforcing mode, and system default policy type. In this case it's permissive, permissive and targeted as shown in the Figure 2.26. When you enabled SELinux the file system is already relabeled. You do not have to select the check box 'relabel on next reboot' once again.
4. Next is the **Boolean** screen. In this screen, you choose individual services and enable or disable (yes or no) specific policy settings. This is where you configure run-time Booleans. For example, for HTTPD Service you may enable allow HTTPD cgi support, allow HTTPD to read home directories and allow HTTPD to support built-in scripting as shown in the Figure 2.27.

Figure 2.26 SELinux Management Tool

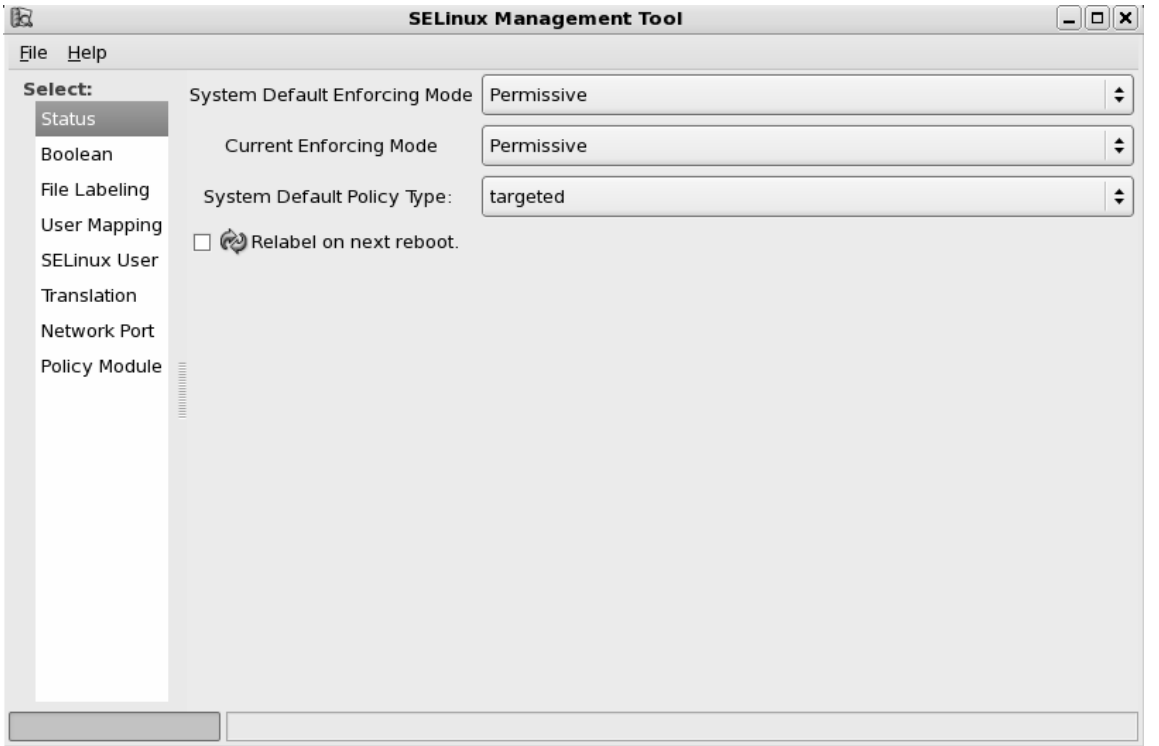
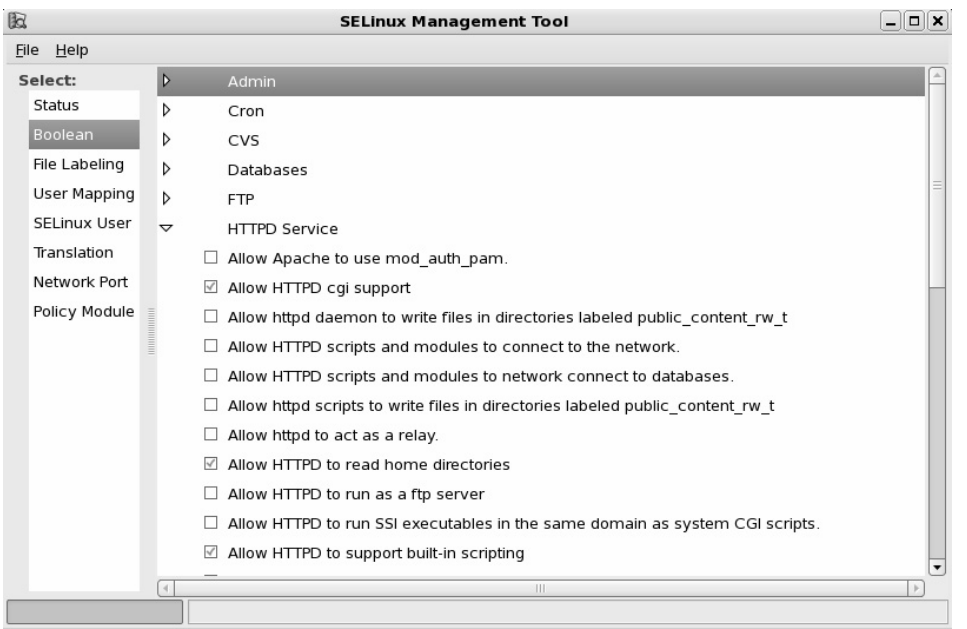
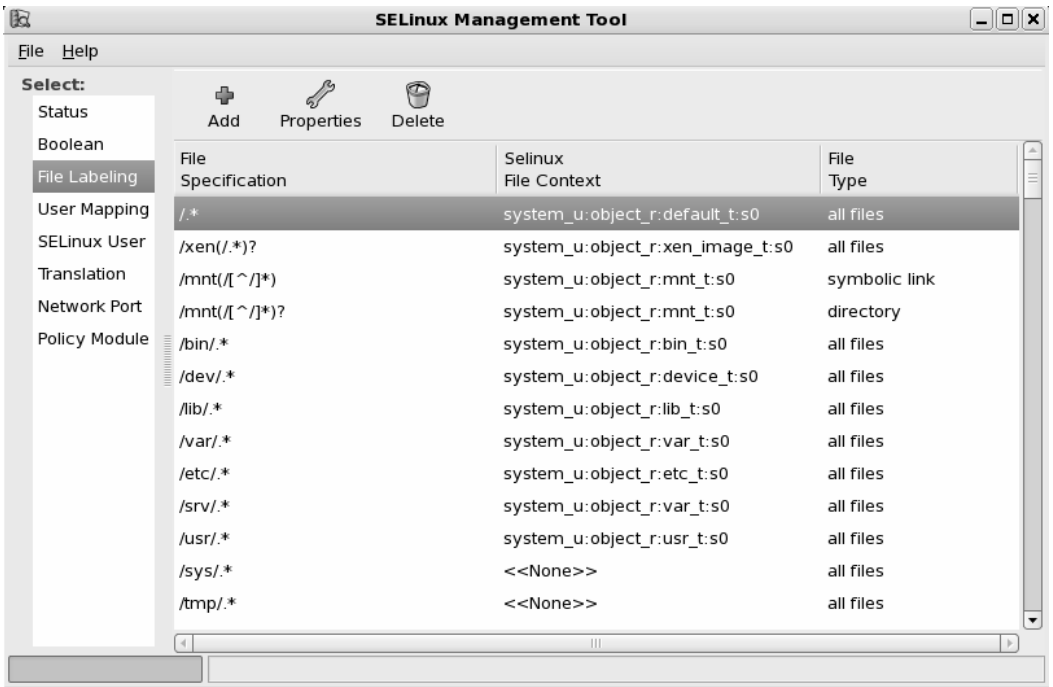


Figure 2.27 Configuring Policy for HTTPD Service using Boolean Tab



5. You may change the file labeling from the **File Labeling** option. You need provide file specification (normally the root, top level directories and mount points), file type (all files, symbolic link, directory, regular file, character device, etc), SELinux Type and MLS (Multi-Level Security such as top secret, secret, confidential and unclassified) details. Click on any row and click on properties to edit the configuration. Figure 2.28 shows File Labeling through SELinux management tool.

Figure 2.28 File Labeling through SELinux Management tool



6. **User Mapping** allows you to categorize the users. For example, files labeled as 'confidential' can only be accessed by the users with a similar categorization (provided the local discretionary access control system also permits the action). This is known as MCS or Multi-Category Security. Figure 2.29 shows user mapping.
7. **SELinux User** option allows you to assign SELinux roles to the SELinux users.
8. **Translation** option allows you to set the sensitivity level translations as shown in Figure 2.30.

Figure 2.29 User Mapping

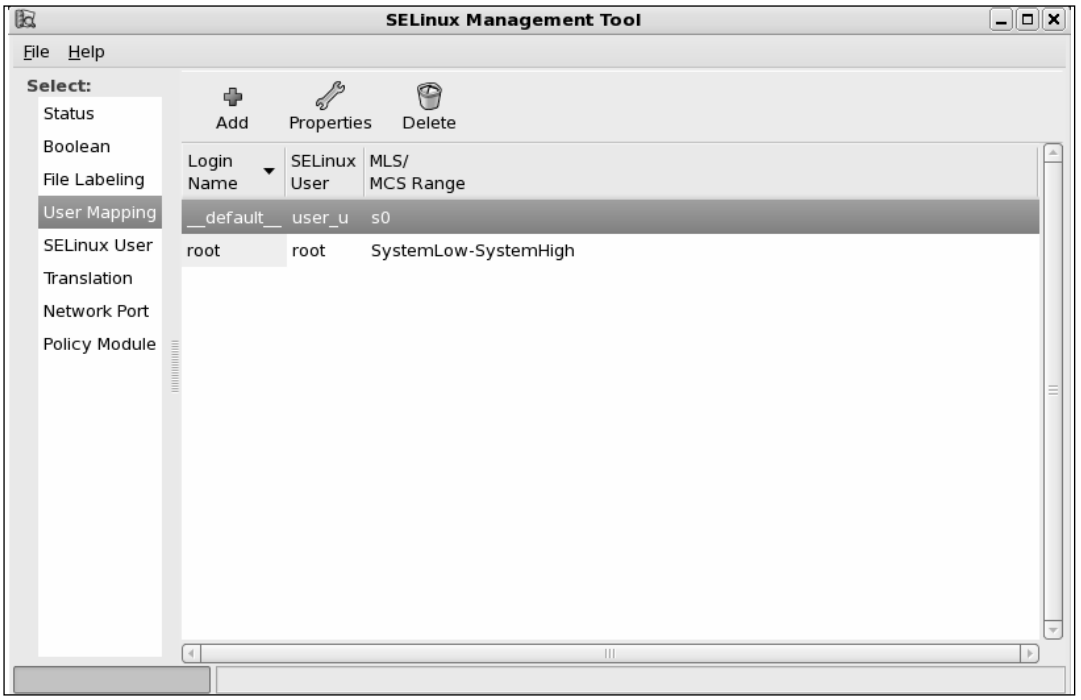
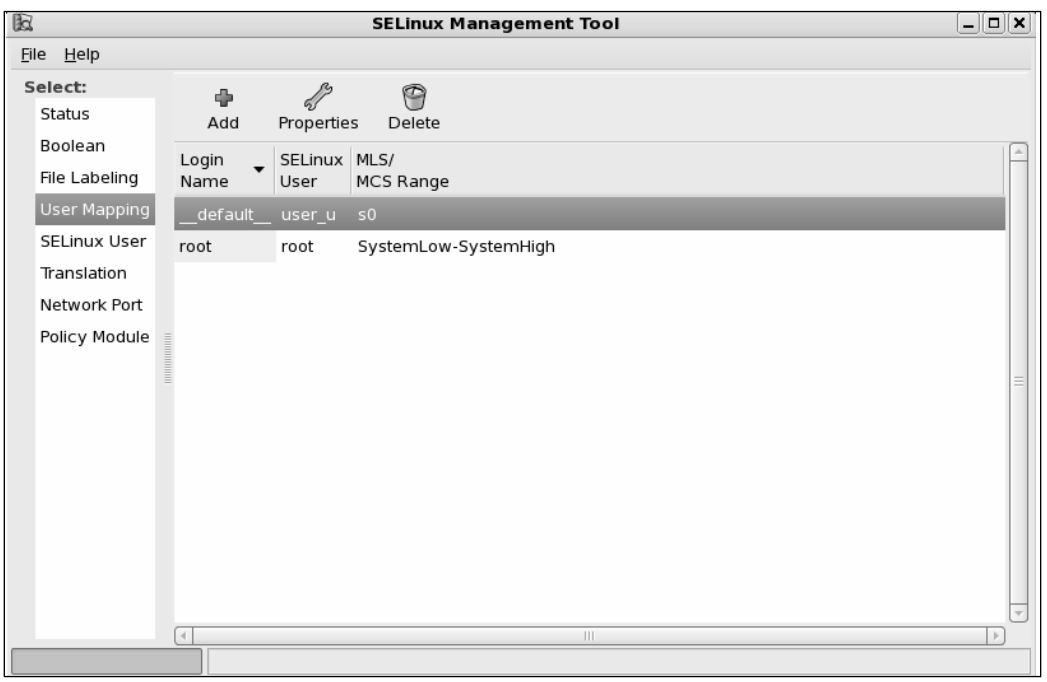
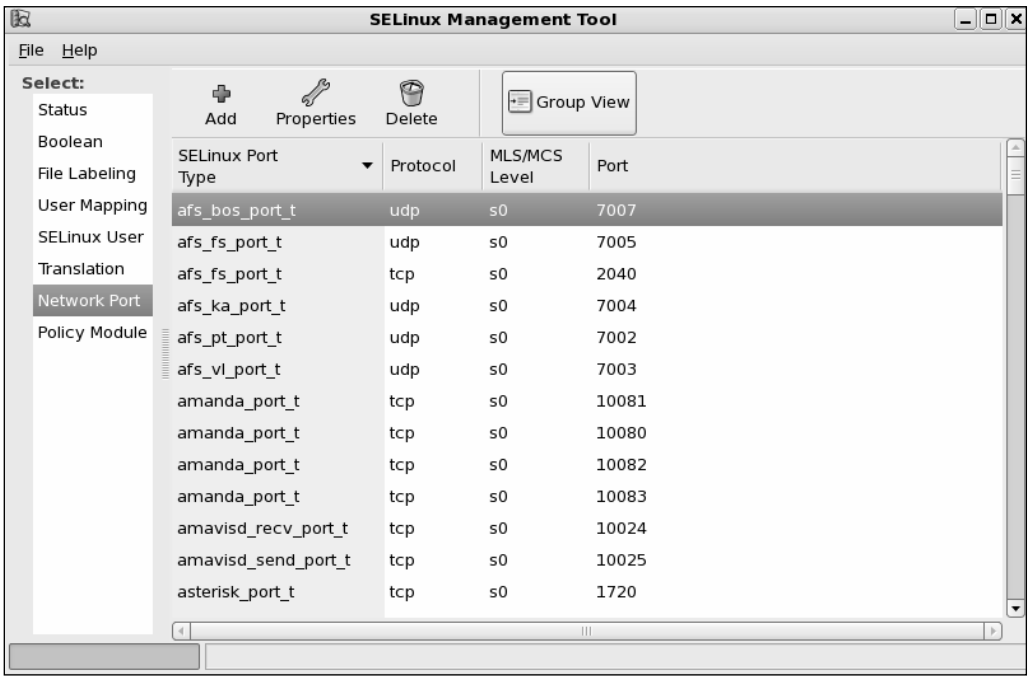


Figure 2.30 Translation



9. **Network Port** option allows you to configure MLS/MCS levels for various network ports. You need to specify the port number, protocol type (tcp or udp), SELinux type and the MLS/MCS Level. You may notice port number 80, protocol tcp, SELinux type http_port_t is configured for MLS/MCS level s0. Figure 2.31 shows Network Port option of SELinux Management tool.

Figure 2.31 Network Port



10. **Policy Module** option allows you to enable or disable audit for specific modules. This will enable additional audit rules that are not reported in the log files.

NOTE

For more information refer to the Red Hat Enterprise Linux 5.0.0 Deployment Guide. To read more about SELinux please visit www.nsa.gov/selinux/.

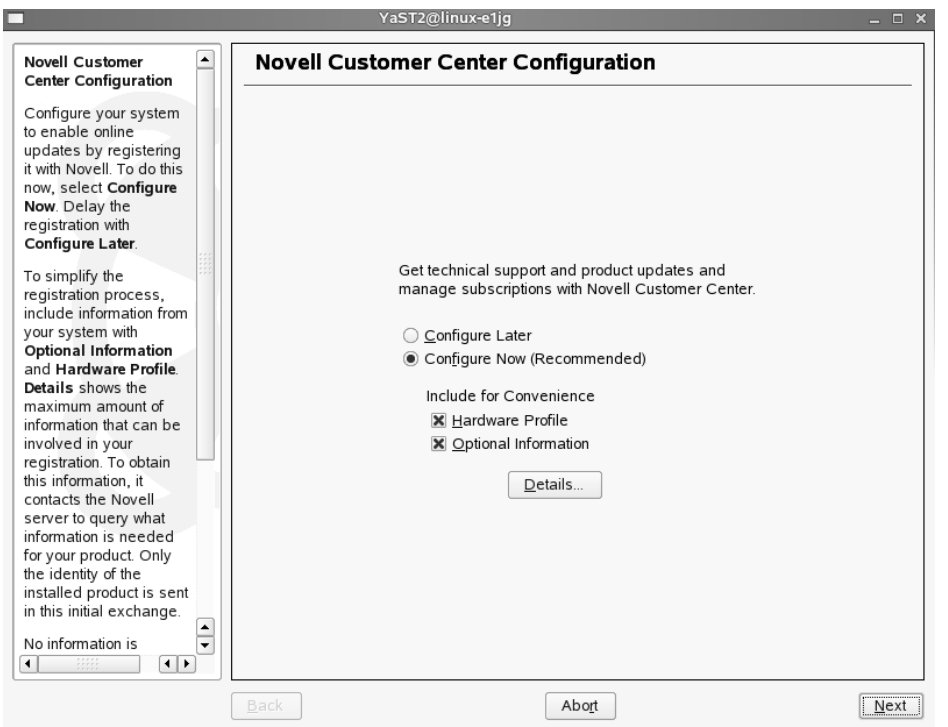
Securing Novell SUSE Linux

SUSE Linux from Novell is another popular vendor supported Linux. SUSE Linux is available for both servers and desktops. Similar to Red Hat Enterprise Linux, SUSE Linux Enterprise Server version 10 provides graphical user interface, easy to install and configure, automatic recognition of variety of hardware, server management and administration tools, automatic updates and technical support.

In this section let us look at the security tasks on Novell SUSE Linux that we have already discussed above on Red Hat Linux.

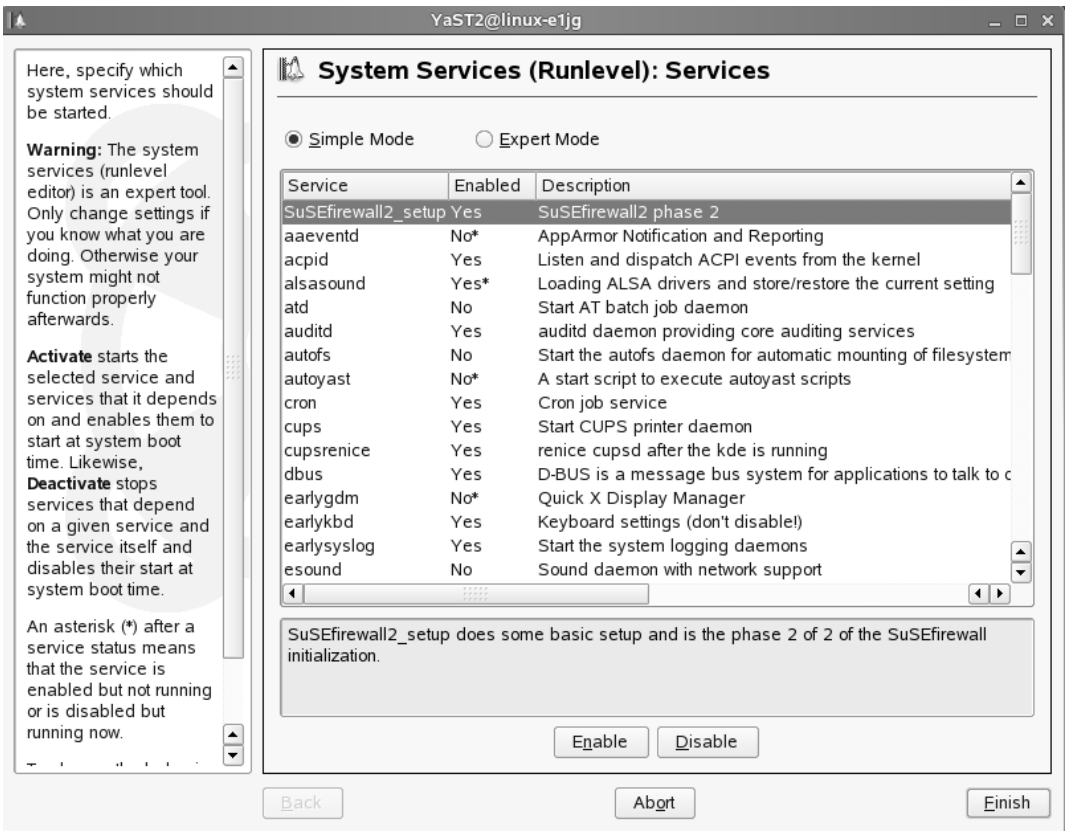
1. **Novell Customer Center** configuration helps you to create the profile of your SUSE Linux server to get the technical support, product updates and manage your subscriptions (licenses). Figure 2.32 shows the Novell customer center configuration.

Figure 2.32 Novell Customer Center Configuration



2. **System Services (Runlevel) Services** tool of YaST (Yet another Setup Tool) allows you to enable or disable your services. Earlier in this chapter we saw how to manually disable the services. You can also edit the run level of individual services. Figure 2.33 shows the service configuration tool.

Figure 2.33 System Services of YaST



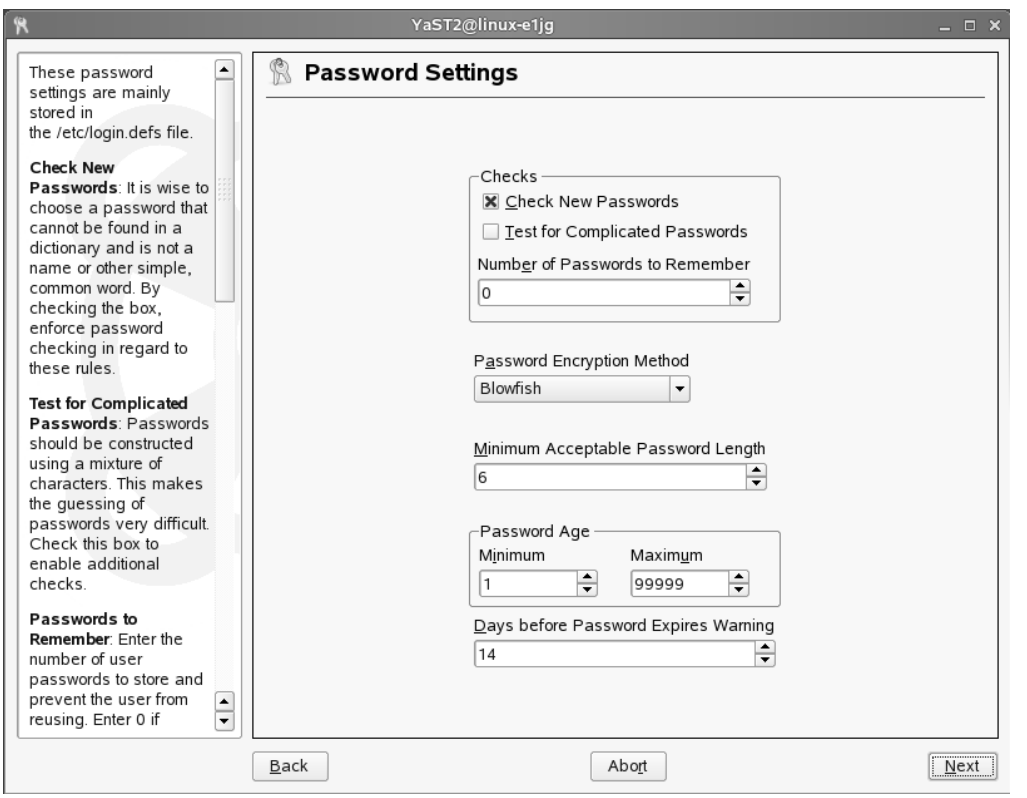
3. Novell SUSE Linux provides you with various graphical user interface tools to configure security settings. To configure local security settings click on **Computer** | **YaST** | **Security and Users** | **Local Security**. Figure 2.34 shows the interface where you can configure password settings. These are some of the settings you performed earlier with Bastille Linux.

   Local Security setting allows you to configure various security settings specific to the server:

4. **System type** options such as home workstation, networked workstation, and network server. When you choose Network Server (network server is the computer that provides services) you will find more options to configure security.
5. **Password settings** such as check for new passwords, test for complicated passwords, number of passwords to remember, password encryption method, and minimum acceptable password length and password age.

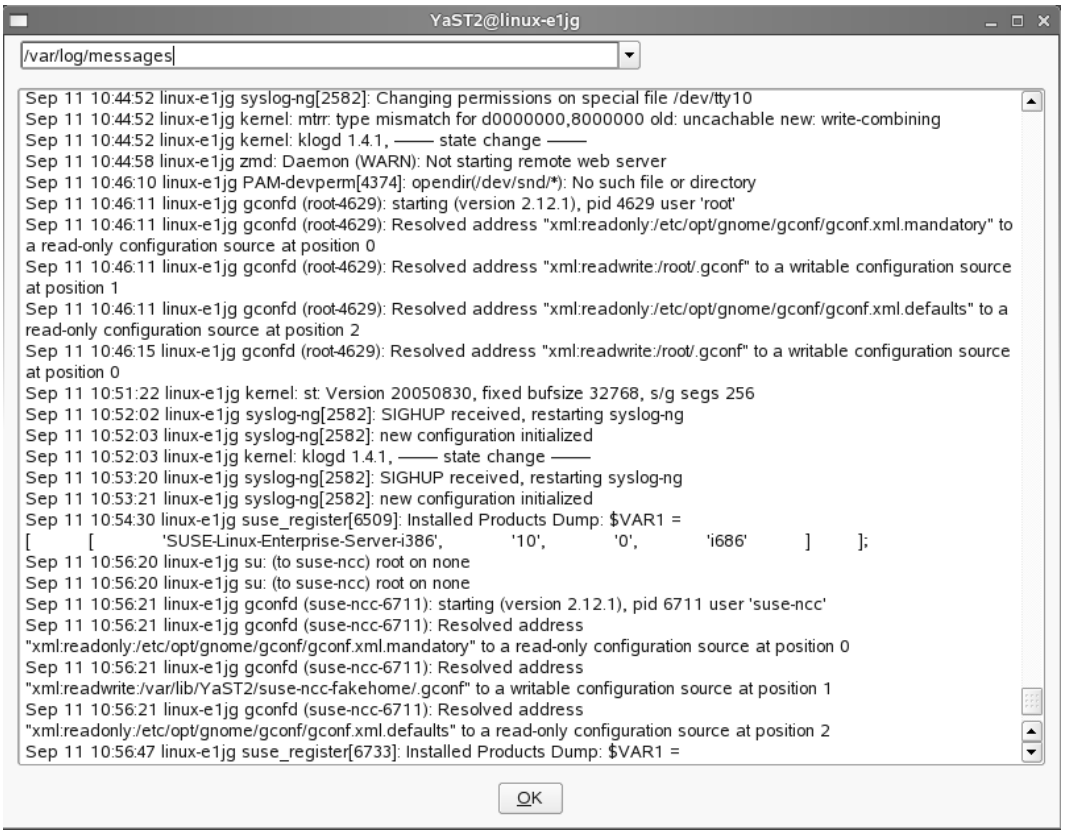
6. **Boot settings** such as interpretation of Ctrl + Alt + Del (options available are ignore, reboot, halt) and shutdown behavior of KDM (only root, all users, nobody and automatic). KDM is the KDE login manager.
7. **Login Security** settings such as delay after incorrect login attempt, record successful login attempts.
8. **User Security** settings such as user ID limitations and group ID limitations (default is 1000 to 60000).
9. **Other Security Settings** such as file permissions (options available are easy, secure, paranoid which is extremely restrictive) and user launching updatedb (options are nobody and root). Updatedb is a program that runs after every boot to update the database with the location information of the files.

Figure 2.34 Local Security Settings



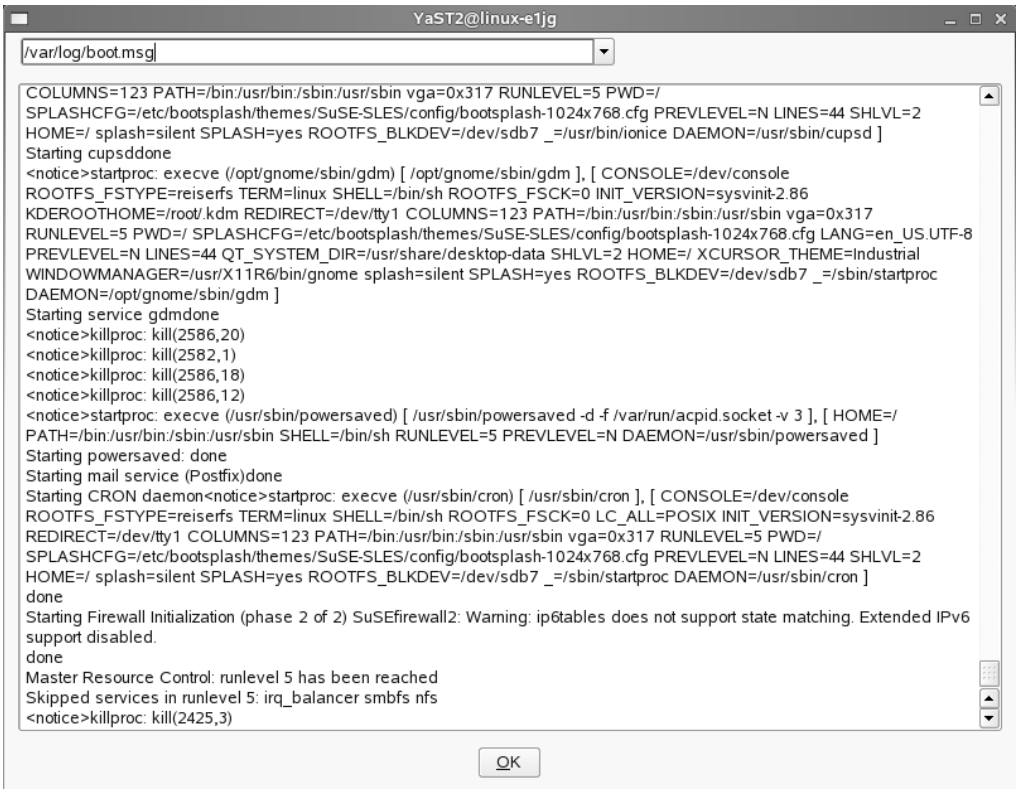
10. **Computer** | **YaST** | **Miscellaneous** | **View System Log**. This will open /var/log/messages. Look for keywords related to the services that you are troubleshooting. Figure 2.35 shows log messages.

Figure 2.35 Logs



11. **Computer** | **YaST** | **Miscellaneous** | **View Start-up Log**. This will open /var/log/boot.msg. This is a good place to locate the errors and status regarding the services that start or fail to start during the boot time. Figure 2.36 shows boot messages.

Figure 2.36 Boot Messages

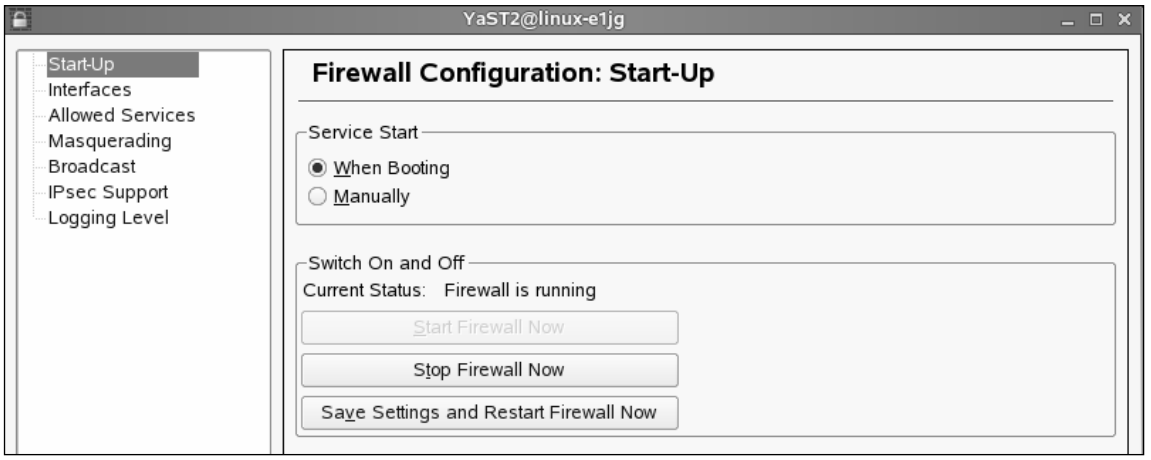


Firewall Configuration

SUSE Linux provides you with a graphical interface to configure firewall on your server. Following section describes the options available to configure firewall. To configure the firewall on your server click on **Computer** | **YaST** | **Security and Users** | **Firewall**.

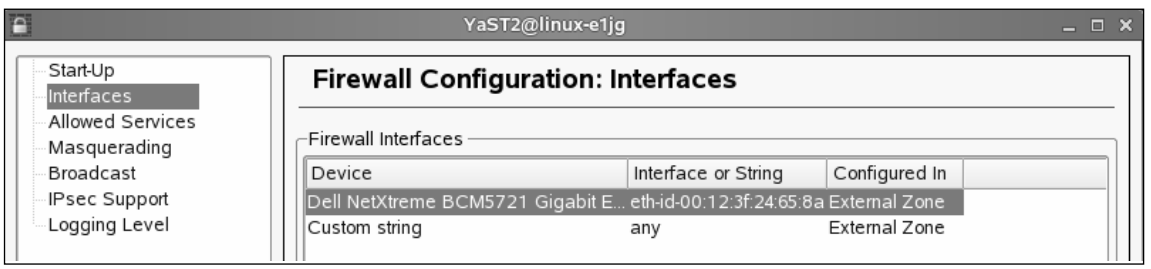
1. To start the firewall when the server starts select '**When Booting'** radio button in the **Start-Up** page. You have the options to start the firewall manually, start or stop the firewall or save the settings to restart the firewall. Figure 2.37 shows the start-up screen of firewall configuration.

Figure 2.37 SUSE Linux Firewall Configuration



2. **Interfaces** participating in the firewall configuration can be configured from **Interfaces** page. In a typical firewall server you may find more than one interface. Figure 2.38 shows Interfaces option of firewall configuration.

Figure 2.38 Firewall Interfaces



3. **Allowed Services** screen allows you to choose the permitted services for every zone. Figure 2.39 shows allowed services page for external zone.You also have an option (not shown in the picture) to enable (checkbox) protection for internal zone. If you could not find the services from the list you can add custom services (ports) manually by specifying the same. Click **Add** to provide tcp, udp, rpc ports and ip protocol options.

Figure 2.39 Allowed Services



4. **Masquerading** option allows you to perform address translation (or address hiding). You need at least one external and one internal interface to configure masquerading.
5. In addition to the above you have options to configure **Broadcast**, **IPsec** support and **Logging** levels.

Novell AppArmor

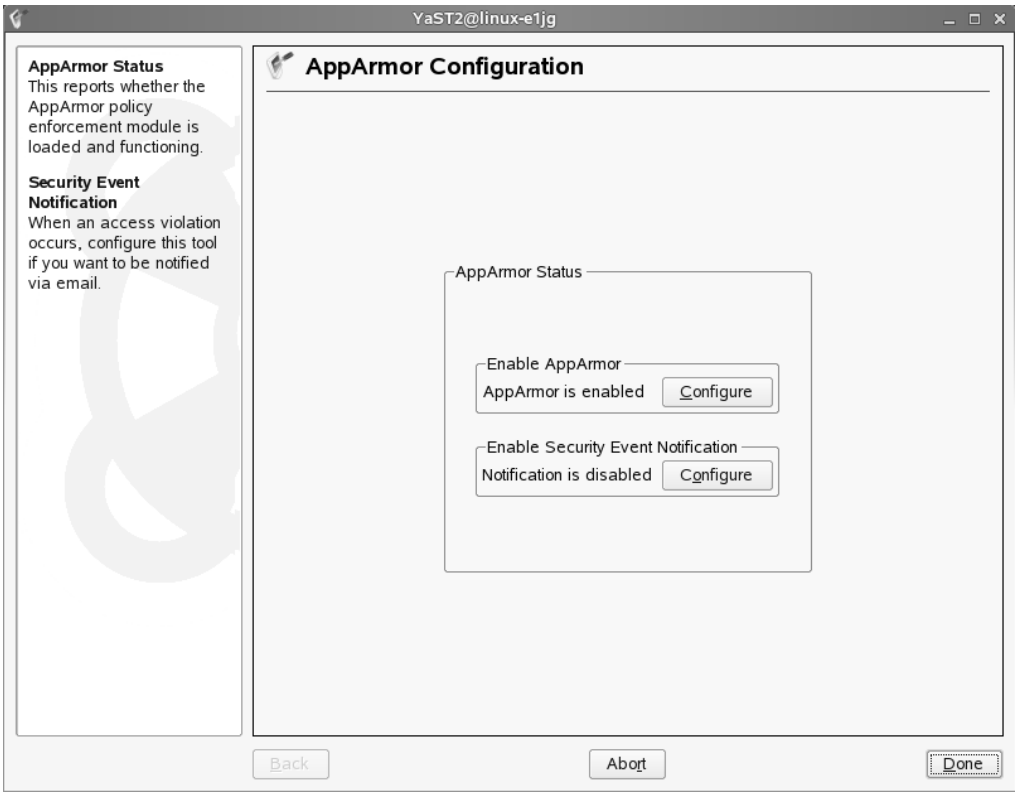
Novell AppArmor provides a mechanism to protect your applications from their vulnerabilities. Novell calls it as '*immunizing.*' To immunize your system you need to install AppArmor (installed by default in Novell SUSE Linux Enterprise Server 10), setup AppArmor profiles and reboot the system. This section briefly describes the process of configuring AppArmor on your SUSE Linux server.

Most of the common application and services have profiles created by default by AppArmor. When you add new applications and services you can create profiles individually for these applications. Make sure you stop the applications before you create profiles. When you create the profile, AppArmor runs *autodep* to analyze and create an initial profile for the application. A manual profiling by you is very much required to avoid restricting the application too granularly that it can not access the directories required to perform its routine functions. AppArmor has a learning mode to monitor various access of the application when you run the same. Information gathered by the learning mode is included in the initial profile created by AppArmor. Creating profiles for critical applications ensures the application run in a specific security environment ensuring no damage to the system due to the vulnerabilities that may get developed time-to-time.

You need to configure AppArmor to immunize programs that may grant privileges, open ports, cron jobs, server daemons and web applications.

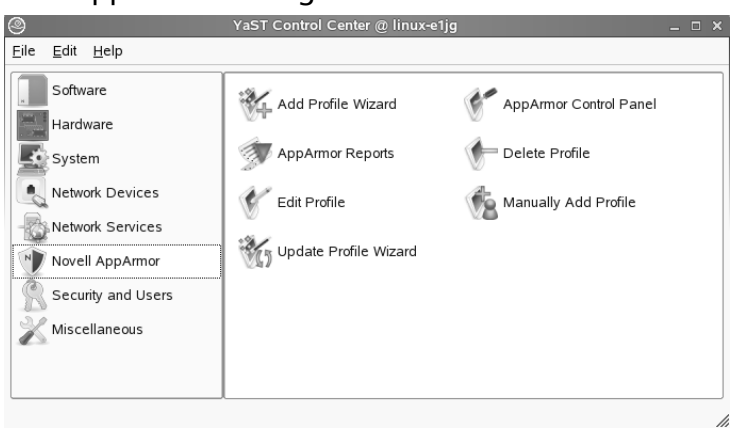
1. First you need to enable AppArmor as shown in the Figure 2.40.

Figure 2.40 Enabling AppArmor



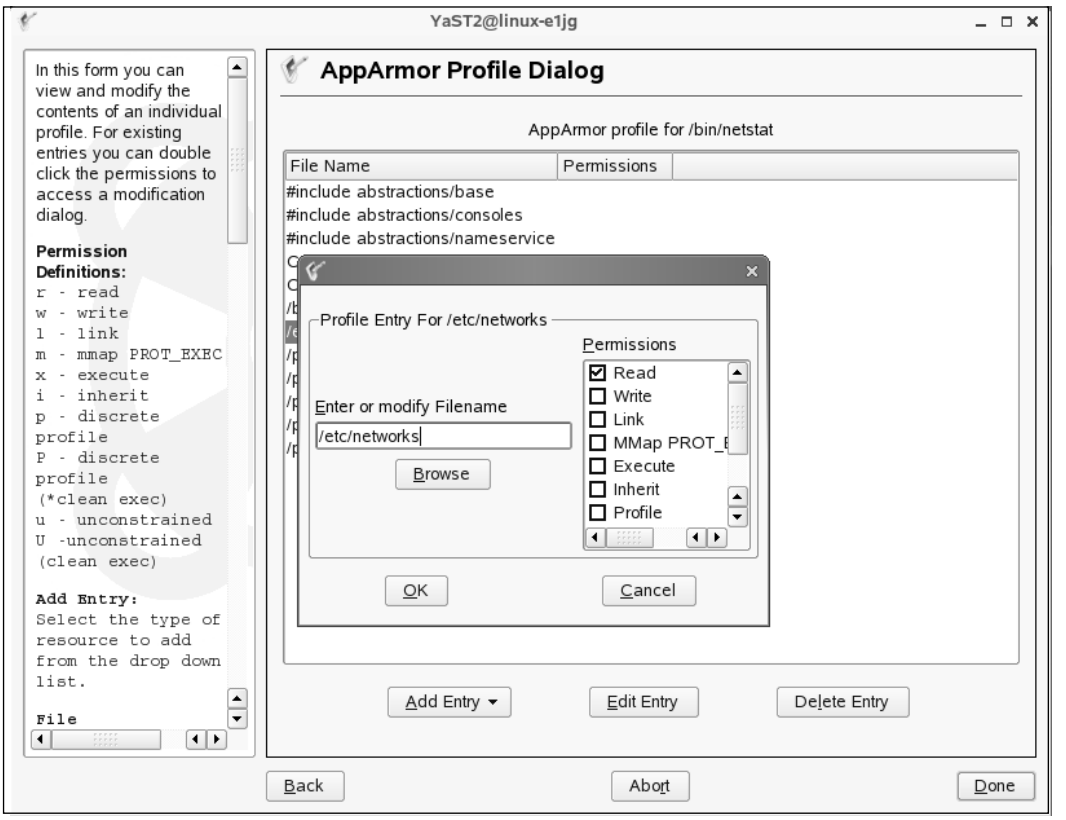
2. Click **Computer** | **YaST** | **Novell AppArmor** as shown in Figure 2.41. You have the following options:
3. Add profile wizard, AppArmor control panel, AppArmor reports, delete profile, edit profile, manually add profile and update profile wizard.

Figure 2.41 Novell AppArmor Configuration



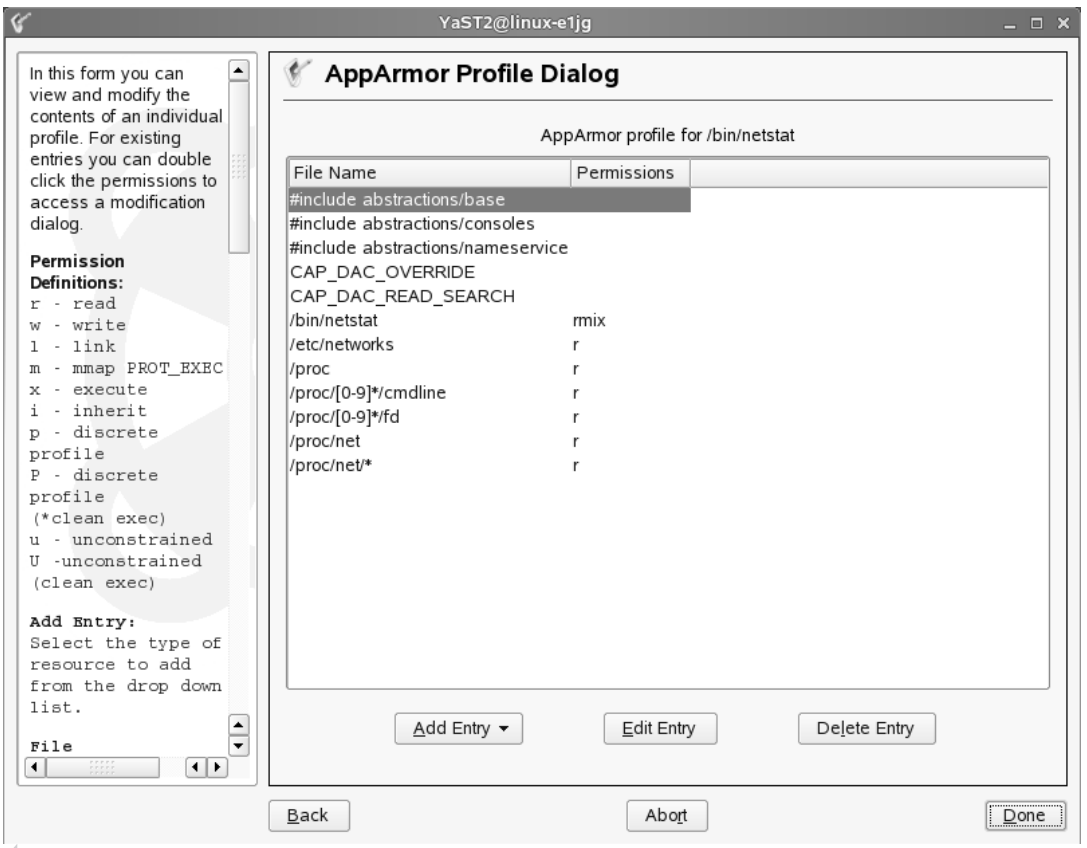
4. Then click on **Edit Profile**. Select the entry for 'netstat' and click on '**Edit Entry**'. You can see the available permissions for the application netstat on /etc/networks file. The options you have are read, write, link, mmap, execute, inherit, discrete profile, unconstrained, Unstrained (clean exec). Figure 2.42 shows AppArmor profile dialog.

Figure 2.42 AppArmor Profile



5. Finally, exit the **Edit Entry** screen. You will see the summary of permissions available for the application netstat to run in your server. Figure 2.43 shows the profile for netstat.

Figure 2.43 AppArmor Profile for Netstat



NOTE

For more information refer to the AppArmor Administration Guide at www.novell.com

Host Intrusion Prevention System

Host Intrusion Prevention System or HIPS as they are popularly known creates a shield around your servers and ensure no malicious attacks or unwarranted changes happen to the system. Apart from providing behavioral rule based protection, signature based analysis and stateful firewall level protection; HIPS also ensure no changes happen to core operating system files from unknown exploits. Commercial grade HIPS provide colorful graphical reporting, advanced alerting system, centralized management and round-the-clock updates

and technical support from the vendors. Many security vendors have come-up with HIPS solutions for Unix/Linux variants. Though the discussion of every HIPS solution or their features is beyond the scope of this book, find below a list of HIPS products available Cisco, McAfee, ISS (now IBM ISS) and Enterasys. Web server and database server specific editions are also available that helps you to prevent attacks such as SQL injection and directory traversal. Needless to say, all these vendors have products for Microsoft platforms as well.

Cisco Security Agent (CSA version 5.2) available for following Linux Server editions apart from Microsoft Windows. CSA is also available for Linux desktop editions.

- Solaris 8 SPARC architecture (64-bit kernel)
- Solaris 9 SPARC architecture (64-bit kernel)
- Red Hat Enterprise Linux 3.0 ES and AS
- Red Hat Enterprise Linux 4.0 ES and AS

Enterasys Dragon Host Sensors are available for the following operating systems:

- Linux
- AIX
- Solaris,
- HP-UX

Enterasys Dragon Host Sensors for Web Intrusion Prevention support:

- WebIPS for Apache with Linux
- WebIPS for Solaris

IBM Internet Security Systems is available for following Unix variants apart from Microsoft Windows:

- IBM RealSecure Server Sensor 7.0 for Solaris
- IBM RealSecure Server Sensor 7.0 for HP-UX
- IBM RealSecure Server Sensor 7.0 for AIX

McAfee Host Intrusion Prevention:

Available platforms include:

- Red Hat Enterprise Linux 4.0 (32-bit only, servers only) supported kernels:
- 2.6.9-11.EL–2.6.9-11.EL-smp
- 2.6.9-22.EL–2.6.9-22.EL-smp
- 2.6.9-34.EL–2.6.9-34.EL-smp

- Sun Solaris (servers only)
- SPARC Solaris 8, sun4u (32-bit or 64-bit kernel)
- SPARC Solaris 9, sun4u (32-bit or 64-bit kernel)
- SPARC Solaris 10, sun4u (64-bit kernel only)

McAfee Host Intrusion Prevention is also available on following web server platforms:

- Apache 1.3.6 and higher
- Apache 2.0.42 or higher
- iPlanet 4.0 and 4.1
- Sun Java System (formerly Sun ONE) 6.0 and 6.1

Linux Benchmark Tools

Center for Internet Security (CIS) is a non-profit organization that helps organizations to mitigate the business risks and stoppage of ecommerce services due to lack of security measures. CIS periodically releases benchmark and scoring tools. Benchmark tools are available for operating systems, network devices and applications.

You need to register before you can download the benchmark and scoring tools. CIS categorizes these tools as level 1 and level 2 tools. Level 1 tool is for system administrations with a basic knowledge of security. These tools are less likely to cause any interruption while run and can be monitored by tools provided by CIS.

CIS tool runs a series of tests on the server or the network devices and produces html and xml reports. The reports provide information on the following areas:

- Patches, Packages and Initial Lockdown
- Minimize xinetd network services
- Minimize boot services
- Kernel Tuning/Network Parameter Modifications
- Logging
- File/Directory Permissions/Access
- System Access, Authentication, and Authorization
- User Accounts and Environment
- Warning Banners
- Reboot

- Anti-Virus Consideration
- Remove Backup Files

In this exercise let's download Red Hat Linux benchmark tool, run the tests and see the results.

1. Download Red Hat Linux benchmark tool from www.cisecurity.org/bench_linux.html. The filename should resemble:

```
ng_scoring_tool-1.0-linux-nojvm.tar
```

2. Download and install Java runtime from www.java.com . Red Hat is currently supported only through a non-JVM (Java Virtual Machine) package. For SUSE Linux you have a bundled jvm package. The Java file name should resemble

```
jre-1_5_0_12-linux-i586.bin
```

3. Execute the .bin file downloaded above to install Java run time.

```
#./ jre-1_5_0_12-linux-i586.bin
```

4. Set the Java environment variables.

   `#set JAVA_HOME=/jre1-1.5.0.12` (the name of the direct created by the above execution file. /jre1-1.5.012/bin is the location where the executables of Java is stored)

```
#export JAVA_HOME
```

5. Install the CIS Red Hat Linux benchmark tool

   `#tar xvf ng_scoring_tool-1.0-linux-nojvm.tar.tar` (this will unzip ng_scoring_tool-1.0-linux-nojvm.jar)

```
#/jre1-1.5.012/bin/java -jar ng_scoring_tool-1.0-linux-nojvm.jar
```

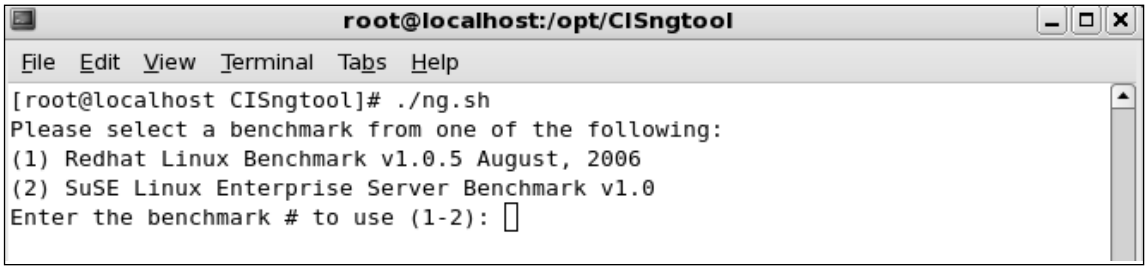
   Follow the simple graphical installation wizard to complete the installation at the default path /opt/CISngtool.

6. To start the tool (as shown in the Figure 2.44)

```
# cd /opt/CISngtool

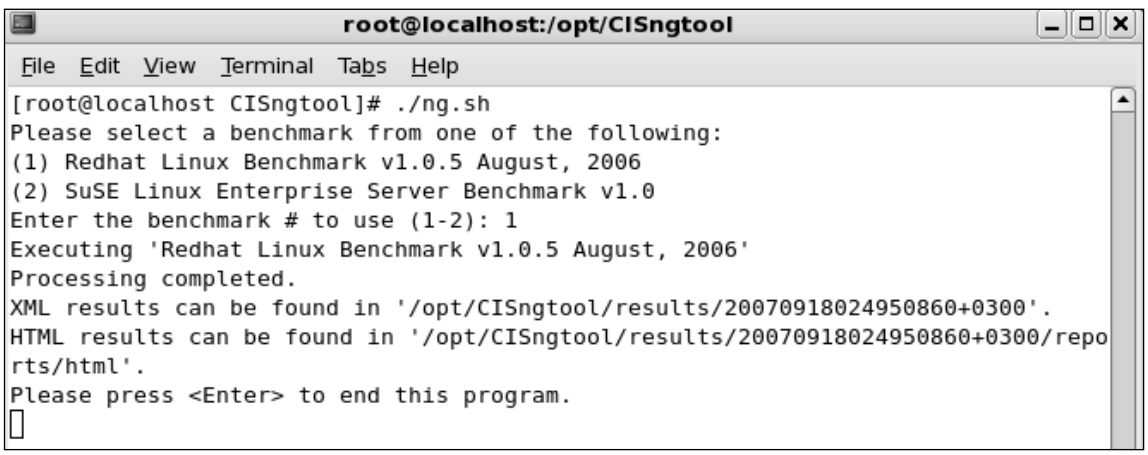
# ./ng.sh
```

Figure 2.44 Executing CIS Red Hat Linux Benchmark tool



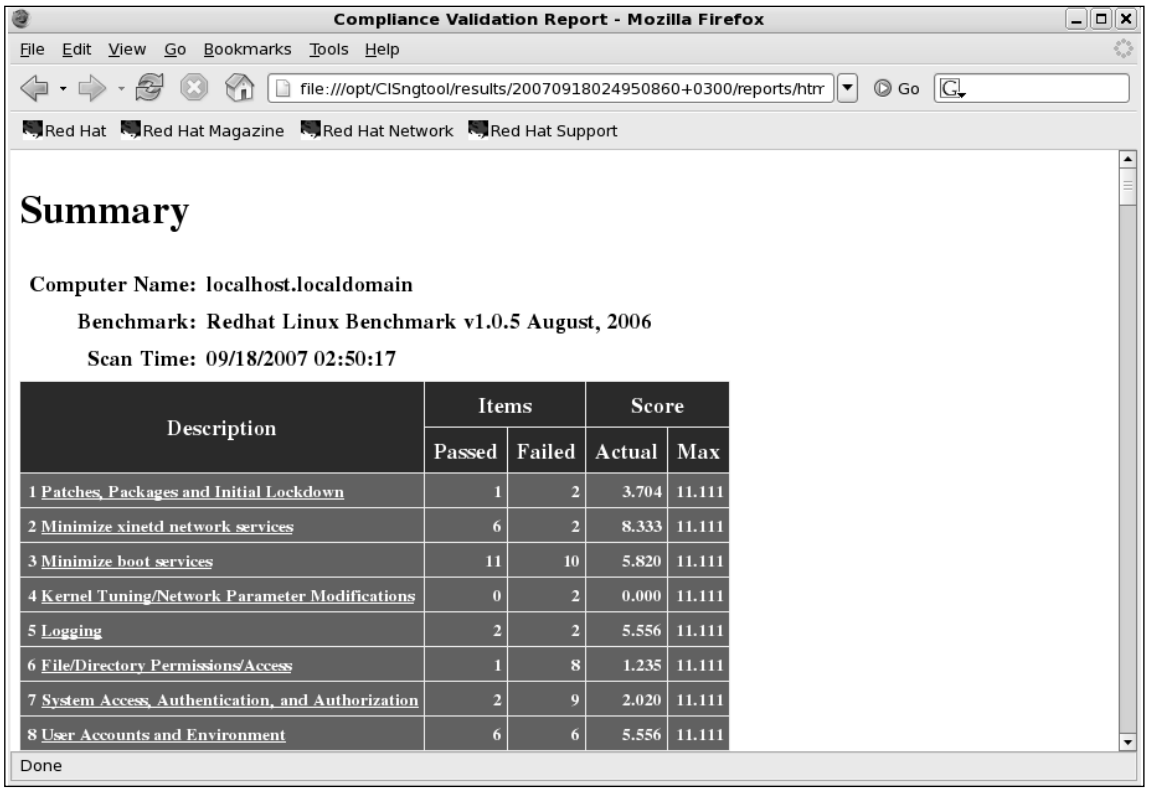
7. **Type 1** and press **Enter** to run the Redhat Linux Benchmark v1.0.5 August, 2006 benchmark tool (as shown in the Figure 2-45). After few minutes of running the tests xml and html results are stored in the /opt/CISngtool/results directory with the date and time stamp as the name of the html file.
8. Press **Enter** to end this program.
9. Open the html file generated by the tool to view the results.

Figure 2.45 Testing Process



10. View the results categorized with details such as passed and failed items, actual and maximum scores along with the name of the server, scan date and time. Figure 2.46 shows the test results.

Figure 2.46 Test Results



Description	Items		Score	
	Passed	Failed	Actual	Max
1 Patches, Packages and Initial Lockdown	1	2	3.704	11.111
2 Minimize xinetd network services	6	2	8.333	11.111
3 Minimize boot services	11	10	5.820	11.111
4 Kernel Tuning/Network Parameter Modifications	0	2	0.000	11.111
5 Logging	2	2	5.556	11.111
6 File/Directory Permissions/Access	1	8	1.235	11.111
7 System Access, Authentication, and Authorization	2	9	2.020	11.111
8 User Accounts and Environment	6	6	5.556	11.111

11. **Click** and **Expand** the view to see the results from the individual categories. Figure 2.47 shows the categories of the results. You may notice the status of individual test results such as not tested, failed or passed.
12. **Click** and further **Expand** the view to read the description of individual tests, for example, **click** 2.1 Disable Standard Services. Figure 2.48 shows the description of the test.

Figure 2.47 Result Categories

Description	Status
1 Patches, Packages and Initial Lockdown	
1.1 Apply Latest OS Patches	Not Tested
1.2 Validate Your System Before Making Changes	Not Tested
1.3 Configure SSH	Failed
1.4 Enable System Accounting	Failed
1.5 Install and Run Bastille	Passed
2 Minimize xinetd network services	
2.1 Disable Standard Services	Failed
2.2 Configure TCP Wrappers and Firewall to Limit Access	Failed
2.3 Only Enable telnet If Absolutely Necessary	Passed
2.4 Only Enable FTP If Absolutely Necessary	Passed
2.5 Only Enable rlogin/rsh/rcp If Absolutely Necessary	Passed

Figure 2.48 Details of Individual Tests

2.1 Disable Standard Services OVAL5 Failed

Description

2.2 Configure TCP Wrappers and Firewall to Limit Access Check Type: OVAL5 Status: Failed

Description

TCP Wrappers and Host-Based Firewalls are presented together as they are similar and complementary in functionality.

CIS also provides downloads for SUSE Linux and Slackware Linux. PDF documents that are downloaded as a part of the benchmark tool archives consist of step-by-step instructions to implement the desired security level on your servers. These recommendations help you to harden your Linux servers. By running the tests periodically you can ensure that your servers are not exposed to serious threats.

NOTE

For more information and to test CIS benchmark tools visit www.cisecurity.org

Summary

This chapter covered the basics of hardening a server to avoid security vulnerabilities using Linux. The main sections covered disabling unnecessary services, locking down ports, Bastille, sudo, and logging enhancers.

It is extremely important to install the latest service pack or updates to the operating system, which fix many security vulnerabilities and bugs before you install any programs. Many services provided with operating systems are not required and can be removed. The key to remember is that the fewer services running, the less potential vulnerability. TCP/UDP ports were covered in this chapter, and how each port is used by specific services. If you block ports on your server, you block the services that use those ports. Locking down ports is an excellent way to reduce exploitations of your system.

Maintaining your server involves downloading service packs and updates, and requires regularly installing bug fixes, security patches, and software updates. These items are available through the operating system vendors, as well as the specific vendors that created the software that you implement.

Bastille is an open source program that facilitates the hardening of a Linux system. It performs many of the tasks listed previously, disabling services and ports that are not required for the system's job functions. Bastille is powerful and can save administrators time from configuring each individual file and program throughout the operating system. Instead, administrators answer a series of "Yes" and "No" questions through an interactive graphical interface. The program automatically implements the administrators' preferences based on the answers to the questions.

Superuser Do (sudo) is an open source security tool that allows an administrator to give specific users or groups the ability to run certain commands as root or as another user. The program can also log commands and arguments entered by specified system users. The developers of sudo state that the basic philosophy of the program is to "give as few privileges as possible, but still allow people to get their work done."

Logging enhancers are tools that simplify logging by allowing logging information to be filtered and often displaying logs in simplified formats. Many open source logging programs exist to make system administration easier. You were introduced in this chapter to SWATCH, scanlogd, and syslog-ng.

SWATCH is an open source package that allows administrators to efficiently monitor system activity. It can monitor events on a system, or a large number of systems, by monitoring system logs for specified events. SWATCH's main function is to monitor messages actively as they are written to log files through the Unix syslog utility.

Scanlogd is an open source program that detects and logs TCP-port scanning on a system. Scanlogd can alert an administrator when the network is being mapped, but it cannot stop the intrusion.

Syslogd-ng is a logging daemon that is the replacement for the traditional syslogd. The "ng" is an acronym for "next generation." The original syslogd was the general Unix logging daemon that handled request for syslog services, but was difficult to configure. Syslogd-ng is easier to configure and offers additional logging features, such as more configurations. For example, syslogd-ng allows administrators to filter messages based on priority, as well as the content of the messages.

Security Enhanced Linux (SELinux) is an initiative from NSA along with the computer security research community to offer enhanced security in the operating systems. SELinux enhances security by utilizing mandatory access control features in Linux. Red Hat offers SELinux from Red Hat Enterprise Linux (RHEL) version 4 onwards.

Novell SUSE Linux is another popular Linux distribution that offers graphical user interface right from the installation, configuration and management. SUSE Linux provides graphical firewall configuration tools as well as to configure finer security settings.

Novell AppArmor protects the system from inherent vulnerabilities of the applications. Novell calls it as *immunizing* the system. By creating AppArmor profiles for open ports, cron jobs and web applications security can be enhanced on Linux servers.

Host Intrusion Prevention System (HIPS) shields the servers from zero-day attacks, providing signature-based, behavior-based protection and ensures core operating system files are not changed by external attacks. Cisco, Enterasys, McAfee, IBM ISS and several other security vendors provide HIPS on major operating system platforms.

Center for Internet Security (CIS) periodically releases benchmarking and scoring tools for operating systems, network devices and applications. You may download these free-of-cost tools and run them on your servers to analyze the security. CIS benchmark tool is available for Red Hat, SUSE and Slackware Linux.

Solutions Fast Track

Updating the Operating Systems

- ☑ Operating system releases usually contain software bugs and security vulnerabilities.
- ☑ Operating system vendors or organizations offer fixes, corrections, and updates to the system. For example, Red Hat offers this material at its Web site, which includes Update Service Packages and the Red Hat Network.

- ☑ You should always ensure your system has the latest necessary upgrades. Many errata and Update Service Packages are not required for every system. You should always read the associated documentation to determine if you need to install it.

Handling Maintenance Issues

- ☑ After your system goes live, you must always maintain it by making sure the most current patches and errata are installed, which include the fixes, corrections, and updates to the system, as well as the applications running on it.
- ☑ You should always check the Red Hat or the appropriate vendor site for the latest errata news and security advisories.
- ☑ For example, Red Hat security advisories provide updates that eliminate security vulnerabilities on the system. Red Hat recommends that all administrators download and install the security upgrades to avoid denial-of-service (DoS) and intrusion attacks that can result from these weaknesses.

Manually Disabling Unnecessary Services and Ports

- ☑ You should always disable vulnerable services and ports on your system that are not used. You are removing risk when you remove unnecessary services.
- ☑ The /etc/xinetd.d directory makes it simple to disable services that your system is not using. For example, you can disable the FTP and Telnet services by commenting out the FTP and Telnet entries in the respective file and restarting the service. If the service is commented out, it will not restart.

Locking Down Ports

- ☑ When determining which ports to block on your server, you must first determine which services you require. In most cases, block all ports that are not exclusively required by these services.
- ☑ To block TCP/UDP services in Linux, you must disable the service that uses the specific port.

Hardening the System with Bastille

- ☑ The Bastille program facilitates the hardening of a Linux system. It saves administrators time from configuring each individual file and program throughout the operating system.

- ☑ Administrators answer a series of "Yes" and "No" questions through an interactive graphical interface. The program automatically implements the administrators' preferences based on the answers to the questions.
- ☑ Bastille can apply restrictive permissions on administrator utilities; disable unnecessary services and ports, and much more.

Controlling and Auditing Root Access with Sudo

- ☑ Sudo (Superuser Do) allows an administrator to give specific users or groups the ability to run certain commands as root or as another user.
- ☑ Sudo features command logging, command restrictions, centralized administration of multiple systems, and much more.
- ☑ The **sudo** command is used to execute a command as a superuser or another user. In order to use the **sudo** command, the user must supply a username and password. If a user attempts to run the command via sudo and that user is not entered in the sudoers file, an e-mail is automatically sent to the administrator, indicating that an unauthorized user is accessing the system.

Managing Your Log Files

- ☑ Logging allows administrators to see who and what has accessed their system. Many helpful Linux log files are located in the /var/log directory.
- ☑ Linux offers commands that allow administrators to access useful log files. Two commands of interest are *last* and *lastlog*. The message file also offers useful data for determining possible security breaches on your system.
- ☑ The Linux logs should be checked frequently to determine if any security violations have occurred on your system. Logs do not offer solutions, so you must analyze the data and decide how to counteract the attack.

Using Logging Enhancers

- ☑ Logging enhancers are tools that simplify logging by allowing logging information to be filtered and often displaying logs in simplified formats.
- ☑ Viewing text-based files with hundreds or thousands of entries can be burdensome, especially if you are only looking for one specific error entry.
- ☑ Three popular logging services used by administrators are SWATCH, scanlogd, and the next generation of syslogd (syslogd-ng).

Security Enhanced Linux

- ☑ Security Enhanced Linux (SELinux) is a research project from NSA and the computer security community supported by major Linux vendors.
- ☑ SELinux enhances security by working on mandatory access control architecture
- ☑ Red Hat Enterprise Linux provides SELinux Management Tool to configure SELinux.

Securing Novell SUSE Linux

- ☑ Novell SUSE Linux is another popular Linux distribution. SUSE Linux comes in server and desktop editions.
- ☑ Novell provides graphical user interface to install, configure and manage Linux servers.
- ☑ Most of the common tasks, configuring OS security settings and firewall can be performed through graphical user interface

Novell AppArmor

- ☑ AppArmor is used to create application specific security profiles.
- ☑ AppArmor performs static analysis to provide an initial profile that consists of permissions required on file system for the application to run smoothly.
- ☑ AppArmor ensures application vulnerabilities do not affect the system.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: I have a server that is strictly a mail server and uses SMTP and POP3. However, I want to download security patches from my vendor's Web site directly to the server. Even though I open the TCP/UDP port 80 (HTTP) and port 53 (DNS), I am unable to download the patches on the mail server.

A: You can download your updates from another system on your network and install the updates on your server through a CD.

Q: Should I place my e-mail server behind the firewall, or in a service network (that is, a "demilitarized zone")?

A: Standard practice is to place the e-mail server in the DMZ. A DMZ is usually comprised of a screening router that blocks most attacks (denial-of-service, system scanning, attacks against Microsoft NetBIOS ports, etc.), and a firewall device that authoritatively blocks incoming traffic, effectively separating the internal network from the world. The DMZ exists between the screening router and the firewall. However, it is often a best practice to place the e-mail server behind the firewall itself. If you do this, however, you must make sure your firewall is configured correctly. Otherwise, a malicious user can take advantage of a misconfigured firewall and gain access to your internal network.

Q: When I install Bastille and run configure, why does the program report that the C compiler cannot create an executable?

A: This error most likely indicates that your system does not have a functioning compiler. If often occurs because you do not have a license, or part of the compiler suite cannot be located. Access and view the config.log to determine the cause. Many compiler components are found in /usr/css/bin. This path may not be identified in the environment variable PATH.

Q: By default, sudo uses syslog(3) for logging. Since I did not change this default during setup, why am I not generating any logging messages?

A: In order to generate sudo log files, you need to create a /var/log/sudo file, and add an entry to the syslog.conf file. Since the default log facility is local2, you must add the following line with TAB keys separating the facility (local2.debug) from the destination (a local logging file).

```
local2.debug                                    /var/log/sudo
```

You must then restart syslogd to ensure that it re-reads the file.

Q: I am tired of entering my password in sudo each time my ticket expires. How can I avoid this hassle?

A: Use the NOPASSWD tag in sudoers for specific users and commands by inserting the tag before the command list. If you want to disable all sudo passwords, there are two methods. You can run configure with the **— without-passwd** option, or you can add **!authenticate** to the Default line in sudoers. Finally, you can disable passwords to users and hosts in sudoers by adding specific user or host Defaults entries. See the sudo man file for specifics on disabling sudo password prompts.

Chapter 3

Enumeration and Scanning Your Network

Solutions in this chapter:

- **Scanning**
- **Enumeration**

Introduction

You now have hardened your system using open source tools. Changes are constantly made to production systems. In addition, malicious users are constantly discovering and exploiting new weaknesses. This chapter discusses ways to scan or *penetration test* your own systems for weaknesses that may already exist or may develop. Penetration testing your own network will help you to see potential weaknesses through the eyes of an attacker, and will help you to close up the holes. You will learn how to scan your hard drive, and then scour running processes to determine ifany problems exist. Finally, you will learn how to deploy the Nessus scanner to test for common server weaknesses. By the end of this chapter, you will have a system that is reasonably secure, and can actually test additional servers for vulnerabilities.

Scanning

During the scanning phase, you will begin to gather information about your network's purpose, specifically what ports (and possibly what services) it offers. Information gathered during this phase is also traditionally used to determine the operating system (or firmware version) of the target devices. The list of active targets gathered from the footprinting phase is used as the target list for this phase. This is not to say that you cannot specifically target *any* host within your approved ranges, but understand that you may lose time trying to scan a system that perhaps does not exist, or may not be reachable from your network location. Often, your penetration tests are limited in time frame, so your steps should be as streamlined as possible to keep your time productive. Put another way; only scan those hosts that appear to be alive, unless you literally have "time to kill."

Enumeration

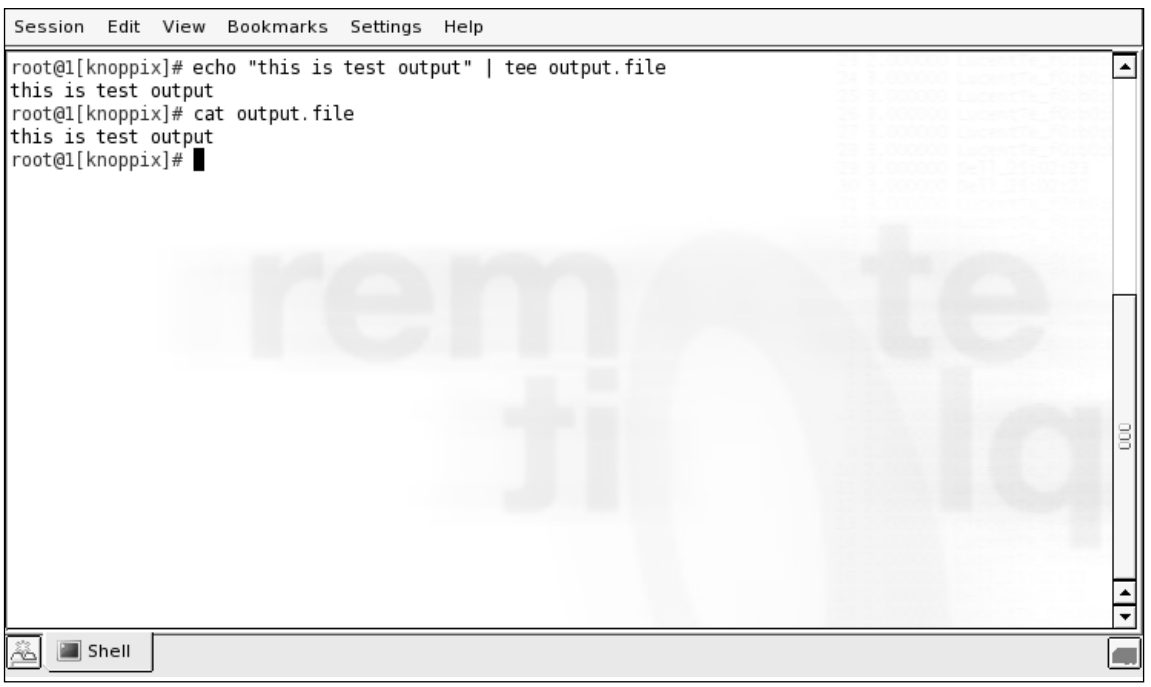
So, what is enumeration? *Enumeration* is a fancy term for listing and identifying the specific services and resources that are offered by a network. You perform enumeration by starting with a set of parameters, like an IP address range, or a specific Domain Name Service (DNS) entry, and the open ports on the system. Your goal for enumeration is a list of services that are known and reachable from your source. From those services, you move further into deeper scanning, including security scanning and testing, the core of penetration testing. Terms such as *banner grabbing* and *fingerprinting* fall under the category of enumeration. The most common tools associated with enumeration include nmap, when run with the *-sV* and *-O* flags, and amap.

An example of successful enumeration would be to start with host 10.0.0.10, and TCP port 22 open. After enumeration, you should be able to state that OpenSSH v3.91 is running with protocol versions 1, 1.5, and 2. Moving into fingerprinting, ideal results would be Slackware Linux v10.1, kernel 2.4.30. Granted, often your enumeration will not get to this

level of detail, but you should still set that as your goal. The more information you have, the better.

Keeping good notes is very important during a pen test, and is especially important during this phase as well. If the tool you are using cannot output a log file, make sure you use tools like `tee`, which will allow you to direct the output of a command to your terminal and to a log file, as demonstrated in Figure 3.1. Sometimes, you may want to know the exact flags or switches you used when you ran a tool, or what the verbose output was.

Figure 3.1 Demonstration of the *tee* Command



You can perform enumeration using either active or passive methods. Proxy methods may also be considered passive, as the information you gather will be from a third source, rather than intercepted from the target itself. However, a truly passive scan should not involve any data being sent from the host system. Active methods are the more familiar in which you send certain types of packets, and then receive packets in return.

Once enumeration is complete, you will have a list of targets that you will use for the next stage, scanning. You need to have specific services that are running, versions of those services, and any host or system fingerprinting that you could determine. Moving forward without this information could hamper your further efforts in exploitation.

How Scanning Works

The list of potential targets fed from the footprinting phase can be expansive. To streamline the scanning process, it makes sense to first determine if the systems are up and responsive. Several methods can be used to test a TCP/IP-connected system's availability, but the most common technique uses Internet Control Message Protocol (ICMP) packets.

Chances are that if you have done any type of network troubleshooting, you will recognize this as the protocol that `ping` uses. The ICMP echo request packet is a basic one that according to Request for Comments (RFC) 1122 every host needs to implement and respond to. In reality, however, many networks, internally and externally, block ICMP echo requests to defend against one of the earliest DoS attack, the ping flood. They may also block it to prevent scanning from the outside.

If ICMP packets are blocked, TCP ACK packets can also be used. This is often referred to as a "TCP ping." RFC 1122 states that unsolicited ACK packets should return a TCP RST. Therefore, sending this type of packet to a port that is allowed through a firewall, such as port 80, the target should respond with an RST indicating that the target is active.

When you combine either ICMP or TCP ping methods to check for active targets in a range, you perform a "ping sweep." Such a sweep should be done and captured to a log file that specifies active machines that you can later input into a scanner. Most scanner tools will accept a carriage return delimited file of IP addresses.

Port Scanning

Although there are many different port scanners, they all operate in much the same way. There are a few basic types of TCP port scans, the most common of which is a SYN scan (or "SYN stealth scan"), named for the TCP SYN flag, which appears in the TCP connection sequence or "handshake." This type of scan begins by sending a SYN packet to a destination port. The target receives the SYN packet, responding with a SYN/ACK response if the port is open, or an RST if the port is closed. This is typical behavior of most scans; a packet is sent, the return is analyzed, and a determination is made about the state of the system or port. SYN scans are relatively fast, and relatively stealthy, since a full handshake does not occur. Since the TCP handshake did not complete, the service on the target does not see a connection, and does not get a chance to log.

Other types of port scans that may be used for specific situations, which we will discuss later in the chapter, would be port scans with various TCP flags set, such as FIN, PUSH, and URG. Different systems respond differently to these packets, so there is an element of OS detection when using these flags, but the primary purpose is to bypass access controls that specifically key on connections initiated with specific TCP flags set. Table 3.1 is a summary of the different nmap options, along with the scan types initiated and expected response.

Table 3.1 nmap Options and Scan Types

nmap Switch	Type of Packet Sent	Response if Open	Response if Closed	Notes
-sT	OS-based connect()	Connection Made	Connection Refused or Timeout	Basic nonprivileged scan type
-sS	TCP SYN packet	SYN/ACK	RST	Default scan type with root privileges
-sN	Bare TCP packet (no flags)	Connection Timeout	RST	Designed to bypass non-stateful firewalls
-sF	TCP packet with FIN flag	Connection Timeout	RST	Designed to bypass non-stateful firewalls
-sX	TCP packet with FIN, PSH, and URG flags	Connection Timeout	RST	Designed to bypass non-stateful firewalls
-sA	TCP packet with ACK flag	RST	RST	Used for mapping firewall rulesets, not necessarily open system ports
-sW	TCP packet with ACK flag	RST	RST	Uses value of TCP Window (positive or zero) in header to determine if filtered port is open or closed
-sM	TCP FIN/ACK packet	Connection Timeout	RST	Works for some BSD systems
-sI	TCP SYN packet	SYN/ACK	RST	Uses a "zombie" host that will show up as scan originator
-sO	IP packet headers	Response in Any Protocol	ICMP Unreachable (Type 3, Code 2)	Used to map out which IP protocols are used by host

Continued

Table 3.1 continued nmap Options and Scan Types

nmap Switch	Type of Packet Sent	Response if Open	Response if Closed	Notes
-b	OS-based connect()	Connection Made	Connection Refused or Timeout	FTP bounce scan used to hide originating scan source
-sU	Blank UDP header	ICMP Unreachable (Type 3, Codes 1, 2, 9, 10, or 13)	ICMP Port Unreachable (Type 3, Code 3)	Can be slow due to timeouts from open and filtered ports
-sV	Sub-protocol specific probe (SMTP, FTP, HTTP, etc.)	N/A	N/A	Used to determine service running on open port, uses service database, can also use banner grab information
-O	Both TCP and UDP packet probes	N/A	N/A	Uses multiple methods to determine target OS/firmware version

Going Behind the Scenes with Enumeration

Enumeration is based on the ability to gather information from an open port, by either straightforward banner grabbing when connecting to an open port, or by inference from the construction of a returned packet. There is not much true magic here, as services are supposed to respond in a predictable manner; otherwise, they would not have use as a service! All of this starts from specified hosts and ports.

Service Identification

Now that the open ports are captured, you need to be able to verify what is running on said ports. You would normally think that SMTP is running on TCP 25, but what if the administrator of the system is trying to obfuscate the service, and is running telnet instead? The easiest way to check the status of a port is a banner grab. Upon connecting to a service, the target's response is captured and compared to a list of known services, such as the response

when connecting to an OpenSSH server as shown in Figure 3.2. The banner in this case is evident, as is the version of the service, OpenSSH version 3.9p1.

Figure 3.2 Checking Banner of OpenSSH Service



RPC Enumeration

Some services are wrapped in other frameworks, such as Remote Procedure Call (RPC). On UNIX-like systems, an open TCP port 111 indicates this. UNIX-style RPC (used extensively by systems like Solaris) can be queried with the *rpcinfo* command, or a scanner can send NULL commands on the various RPC-bound ports to enumerate what function that particular RPC service performs.

Fingerprinting

The goal of system fingerprinting is to determine the operating system version and type. There are two common methods of performing system fingerprinting: active and passive scanning. The more common active methods use responses sent to TCP or ICMP packets. The TCP fingerprinting process involves setting flags in the header that different operating systems and versions respond to differently. Usually, several different TCP packets are sent and the responses are compared to known baselines (or fingerprints) to determine the remote OS. Typically, ICMP-based methods use fewer packets than TCP-based methods, so when you need to be more stealthy and can afford a less-specific fingerprint, ICMP may be

the way to go. Higher degrees of accuracy can be achieved by combining TCP/UDP and ICMP methods, assuming that no device between you and the target is reshaping packets and mismatching the signatures.

For the ultimate in stealthy detection, passive fingerprinting can be used. Similar to the active method, this style of fingerprinting does not send any packets, but relies on sniffing techniques to analyze the information sent in normal network traffic. If your target is running publicly available services, passive fingerprinting may be a good way to start your fingerprinting. A drawback of passive fingerprinting is that it is less accurate than a targeted active fingerprinting session and relies on an existing traffic stream.

Open Source Tools

Now that we've described some of the theories, it is time to implement them with the open source tools. We'll look at several different tools, broken into two categories: scanning and enumeration.

Scanning

We'll begin by discussing tools that aid in the scanning phase of an assessment. Remember, these tools will scan a list of targets in an effort to determine which hosts are up, and what ports and services are available.

Fyodor's nmap

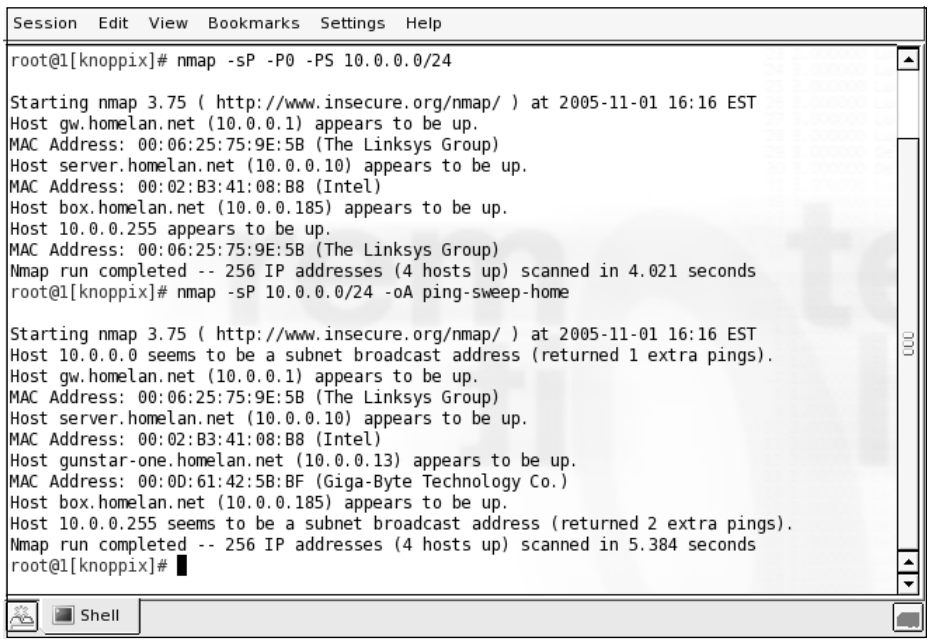
Port scanners accept a target or a range as input, send a query to specified ports, and then create a list of the responses for each port. The most popular scanner is *nmap* written by Fyodor, available from www.insecure.org. Fyodor's multipurpose tool has become a standard item among pen testers and network auditors. While it is not our intent to teach you all the ways to use nmap, we will focus on a few different scan types and options to make the best use of your scanning time and to return the best information.

nmap: Ping Sweep

Before scanning active targets, consider using the *ping sweep* functionality of nmap with the *-sP* option. This option will not port scan a target, but will simply report which targets are up. When invoked as root with *nmap -sP ip_address*, nmap will send *both* ICMP echo packets and TCP SYN packets to determine if a host is up. However, if you know that ICMP is blocked, and don't want to send those unnecessary ICMP packets, you can simply modify nmap's ping type with the *-P* option. For example, *-P0 -PS* enables a TCP ping sweep, with *-P0* indicating "no ICMP ping" and *-PS* indicating "use TCP SYN method." By isolating the scanning method to just one variant, you increase the speed as well, which may not be a big issue when scanning a handful of systems, but when scanning multiple /24 networks, or

even a /16, you may need this extra time for other testing. This is demonstrated in Figure 3.3, where both TCP-only and TCP/ICMP sweeps of a class C network complete in 4.021 seconds and 5.384 seconds, respectively.

Figure 3.3 nmap TCP Ping Scan



nmap: ICMP Options

If nmap can't see the target, it won't scan it unless the *-P0* (do not ping) option is used. Using the *-P0* option can create problems since nmap will scan each of the target's ports, even if the target isn't up, which can waste time. To strike a good balance, consider using the *-P* option to select another type of ping behavior. For example, the *-PP* option will use ICMP timestamp requests, and the *-PM* option will use ICMP netmask requests. Before you perform a full sweep of a network range, it might be useful to do a few limited tests on known IP addresses, such as Web servers, DNS, and so on, so you can streamline your ping sweeps and reduce the number of total packets sent and the time taken for the scans.

nmap: Output Options

Capturing the results of the scan is extremely important, as you will be referring to this information later in the testing process. The easiest way to capture all the needed information is to use the *-oA* flag, which outputs scan results in three different formats simultaneously: plain text (.nmap), greppable text (.gnmap), and XML (.xml). The gnmap format is

especially important to note, because if you need to stop a scan and resume it later, nmap will require this file to continue by using the —*resume* switch.

nmap: Stealth Scanning

For any scanning you perform, it is not a good idea to use a connect scan (*-sT*), which fully establishes a connection to a port. Excessive port connections can cause a DoS to older machines, and will definitely raise alarms on any IDS system. Therefore, you should use a stealthy port testing method with nmap, such as a SYN scan. To launch a SYN scan from nmap, you use the *-sS* flag, which produces a listing of the open ports on the target, and possibly open/filtered ports if the target is behind a firewall. The ports returned as open are listed with what service that port corresponds to, based on IANA port registrations, as well as any commonly used ports, such as 31337 for Back Orifice.

In addition to lowering your profile with half-open scans, you may also consider the ftp or "bounce" scan and idle scan options that can mask your IP from the target. The ftp scan takes advantage of a feature of some FTP servers, which allow anonymous users to proxy connections to other systems. If you find during your enumeration that an anonymous FTP server exists, or one to which you have login credentials, try using the *-b* option with *user:pass@server:ftpport*. If the server does not require authentication, you can skip the *user:pass*, and unless FTP is running on a nonstandard port, you can leave out the *ftpport* option as well. The idle scan, using *-sI zombiehost:port*, has a similar result, but a different method of scanning. This is detailed further at Fyodor's Web page (www.insecure.org/nmap/idlescan.html), but the short version is that if you can identify a target with low traffic and predictable IPID values, you can send spoofed packets to your target, with the source set to the idle target. The result is that an IDS sees the idle scan target as the system performing the scanning, keeping your system hidden. If the idle target is a trusted IP address and can bypass host-based access control lists (ACLs), even better! Do not expect to be able to use a bounce or idle scan on every penetration test engagement, but keep looking around for potential targets. Older systems, which do not offer useful services, may be the best targets for some of these scan options.

nmap: OS Fingerprinting

You should be able to create a general idea of the remote target's operating system from the services running and the ports open. For example, ports 135, 137, 139, or 445 often indicate a Windows-based target. However, if you want to get more specific, you can use nmap's *-O* flag, which invokes nmap's fingerprinting mode. Care needs to be taken here as well, as some older operating systems such as AIX prior to 4.1 and older SunOS versions have been known to die when presented with a malformed packet. Keep this in mind before blindly using *-O* across a Class B subnet. Figures 3.4 and 3.5 show the output from a fingerprint scan using *nmap -O*. Note that the fingerprint option without any scan types will invoke a SYN scan, the equivalent of *–sS*, so ports can be found for the fingerprinting process can occur.

Figure 3.4 nmap OS Fingerprint of Windows XP SP2 System

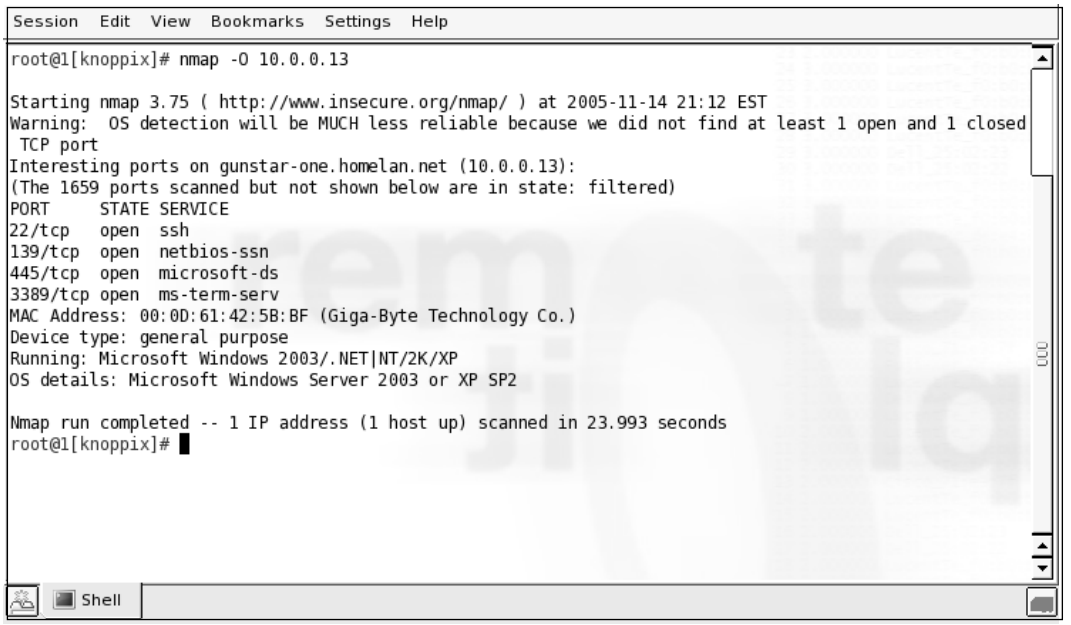
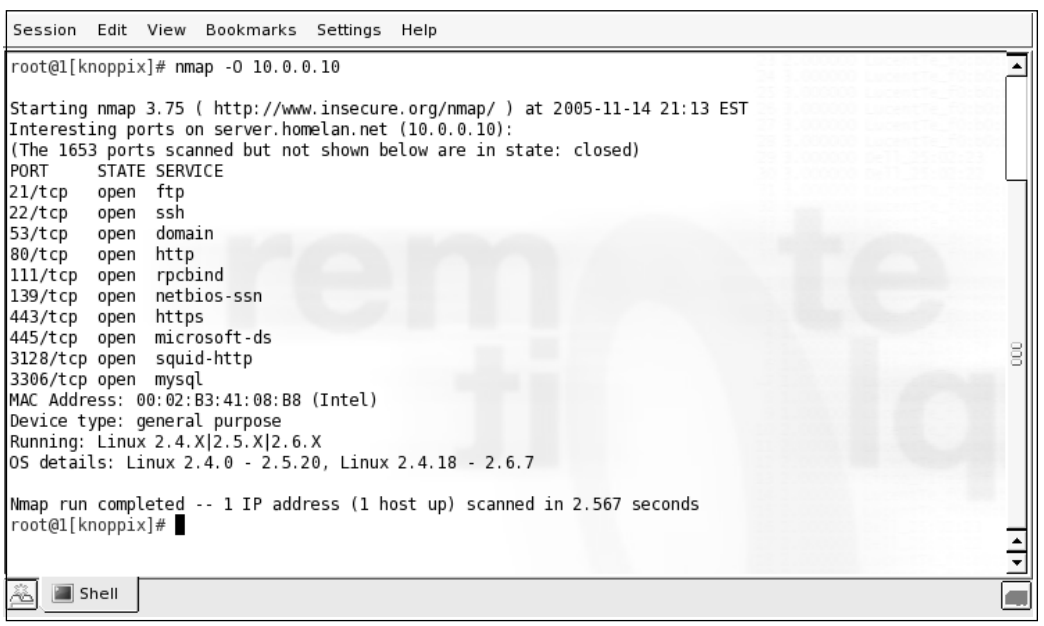


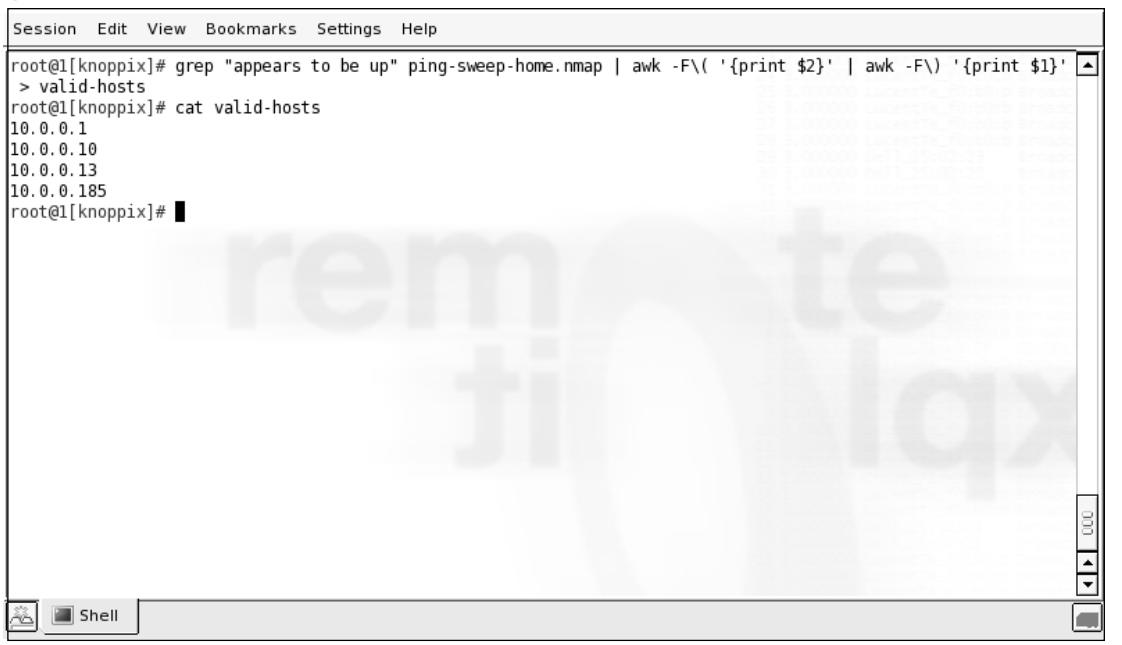
Figure 3.5 nmap OS Fingerprint of Fedora Core 3 Linux System



nmap: Scripting

When you specify your targets for scanning, nmap will accept specific IP addresses, address ranges in CIDR format, and ranges using 192.168.1.100-200 style notation. If you have a host file, which may have been generated from your ping sweep earlier (hint, hint), you can specify it as well, using the *-iL* flag. There are other, more formal nmap parsing programs out there, but Figure 3.6 shows how *awk* can be used to create a quick and dirty hosts file from an nmap ping sweep. Scripting can be very powerful additive to any tool, but remember to check all the available output options before doing too much work, as some of the heavy lifting may have been done for you.

Figure 3.6 awk Parsing of nmap Results File



nmap: Speed Options

nmap allows the user to specify the "speed" of the scan, or the amount of time from probe sent to reply received, and therefore how fast packets are sent. On a fast LAN, you can optimize your scanning by setting the *-T* option to 4, or Aggressive, usually without dropping any packets during send. If you find that a normal scan is taking very long due to ingress filtering, or a firewall device, you may want to enable Aggressive scanning. If you know that an IDS sits between you and the target, and you want to be as stealthy as possible, then using *-T0* or Paranoid should do what you want; however, it will take a long time to finish a scan, perhaps several hours, depending on your scan parameters.

By default, nmap 3.75 with Auditor scans 1663 ports for common services, which will catch most open TCP ports out there. However, sneaky sysadmins may run ports on uncommon ports, practicing security through obscurity. Without scanning those uncommon ports, you may be missing these services. If you have time, or suspect that a system may be running other services, run nmap with the *–p1-65535* parameter, which will scan all 65k TCP ports. Even on a LAN with responsive systems, this will take anywhere from 30 minutes to a few hours. Performing a test like this over the Internet may take even longer, which will also allow more time for the system owners, or watchers, to note the excessive traffic and shut you down.

netenum: Ping Sweep

If you have a need for a very simple ICMP ping sweep program that you can use for scriptable applications, netenum might be useful. It performs a basic ICMP ping and then replies with only the reachable targets. One quirk about netenum is that it requires a timeout to be specified for the entire test. If no timeout is specified, it outputs a CR-delimited dump of the inputted addresses. If you have tools that will not accept a CIDR formatted range of addresses, you might use netenum to simply expand that into a listing of individual IP addresses. Figure 3.7 shows the basic use of netenum in ping sweep mode and in network address expansion mode.

Figure 3.7 netenum

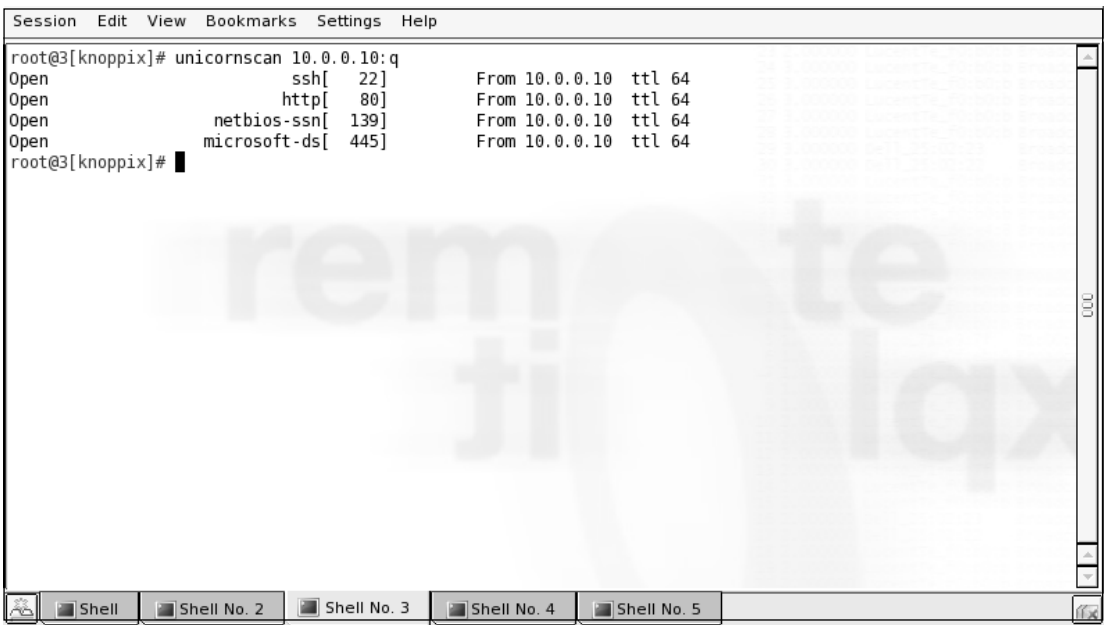


unicornscan: Port Scan

unicornscan is different from a standard port scanning program, and allows you to specify more information, such as source port, packets per second sent, and randomization of source IP information, if needed. For this reason, it may not be the best choice for initial port scans,

but rather more suited for later "fuzzing" or experimental packet generation and detection. Figure 3.8 shows unicornscan in action, performing a basic SYN port scan. unicornscan might be better suited for scanning during an IDS test, where the packet forging capabilities could be put to more use. This is unless you have some "throwaway" IP addresses that you can use for an external test; if you do, then forge away!

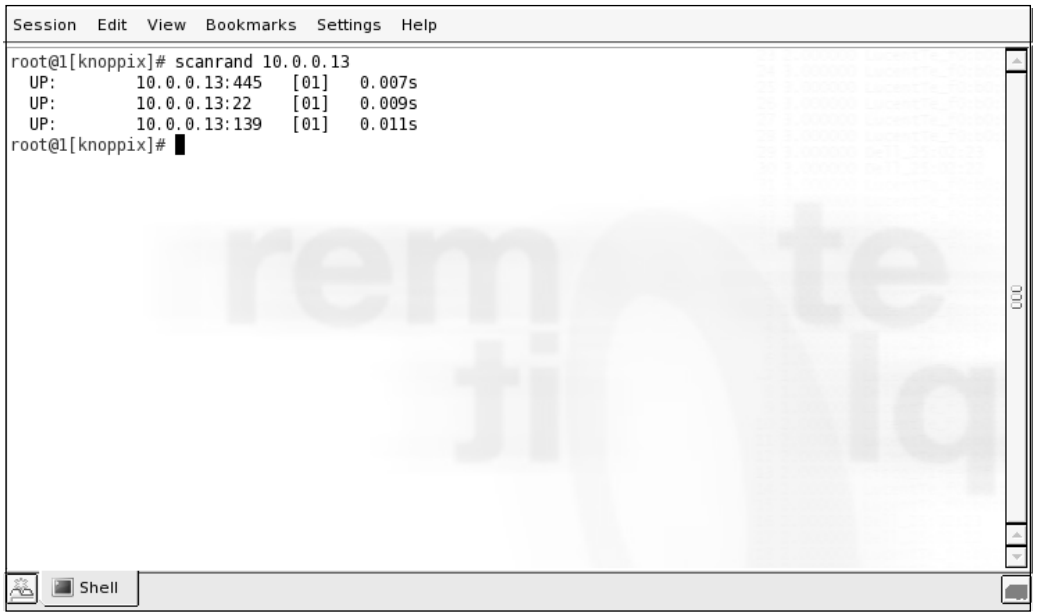
Figure 3.8 unicornscan



scanrand: Port Scan

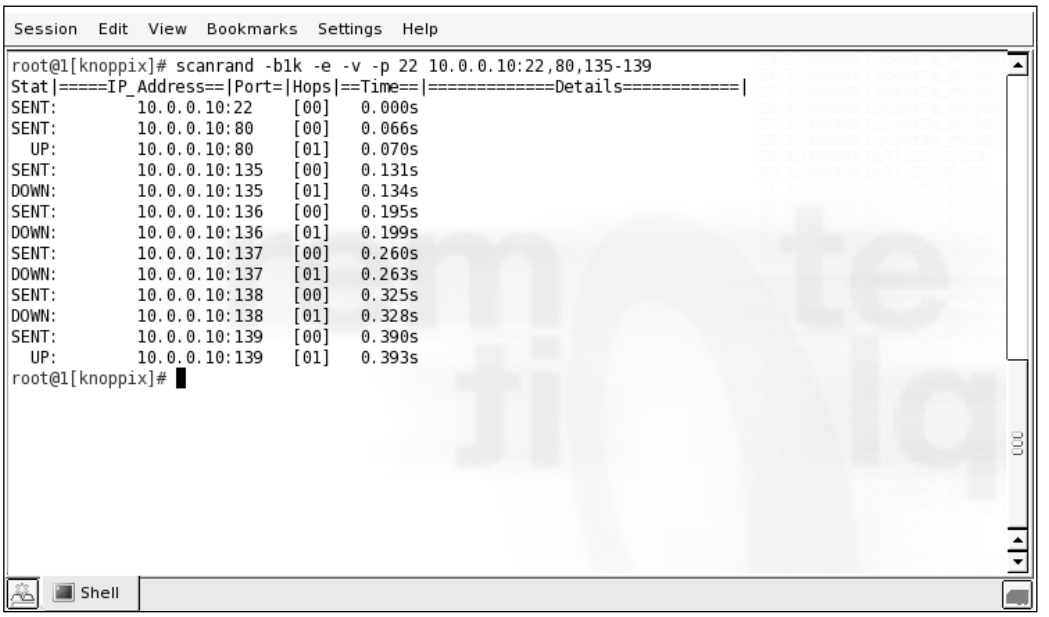
Similar to unicornscan, scanrand offers different options than a typical port scanner does. It implements two separate scanner processes, one for sending requests and one for receiving those requests. Because of this separation, the processes can run asynchronously, which gives a boost in speed. The packets are encoded with digital signatures that allow the processes to keep track of the requests and prevent forged responses from giving false data. Figure 3.9 is a demonstration of scanrand's basic scanning capability.

Figure 3.9 scanrand



Another nice feature of scanrand is the ability to specify bandwidth usage for the scan, from bytes to gigabytes. When performing testing over a very limited connection, such as satellite, the capability to throttle these attempts is very important. In Figure 3.10, scanrand is run using the *-b1k* switch, which limits bandwidth usage to 1 KByte/sec, which is very reasonable for slower connections, even those with relatively high latency. The source port of the scan is set to TCP 22, with the *–p 22* switch, and both open and closed ports are shown using the *-e* and *-v* options.

Figure 3.10 scanrand Limited Bandwidth Testing



Enumeration

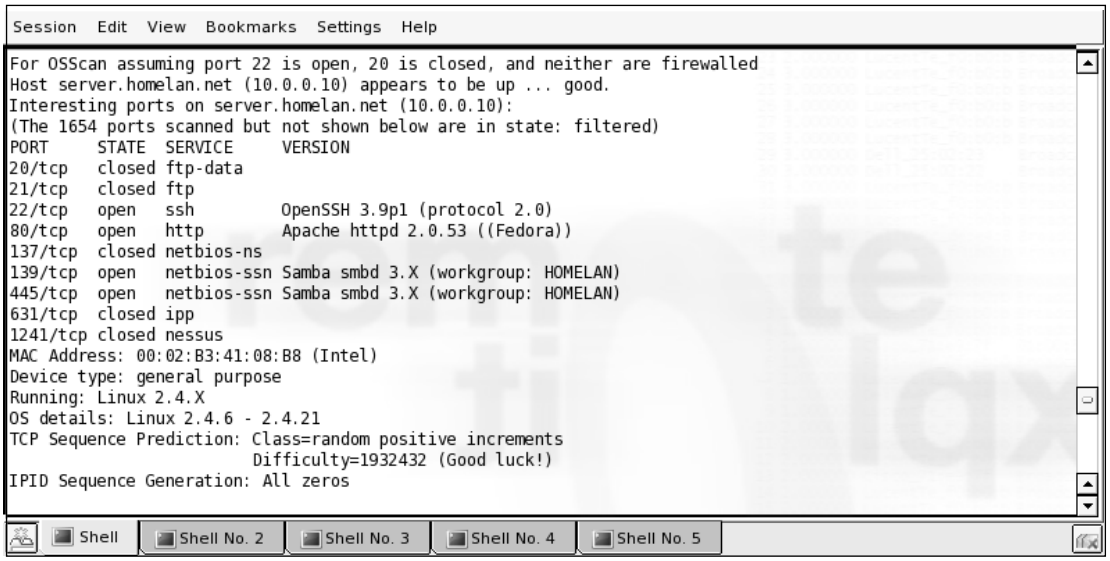
Next, we'll discuss tools that aid in the enumeration phase of an assessment. Remember, these tools will scan a list of targets and ports to help determine more information about each target. The enumeration phase usually reveals program names, version numbers, and other detailed information that will eventually be used to determine vulnerabilities on those systems.

nmap: Banner Grabbing

The version-scanning feature of nmap is invoked with the *-sV* flag. Based on a returned banner, or on a specific response to an nmap-provided probe, a match is made between the service response and the nmap service fingerprints. This is a newer feature and since it interrogates discovered services, many IDS vendors will be writing signature files for this type of behavior, so use it with caution.

Figure 3.11 shows a successful scan using nmap *–sS –sV –O –v* against a Linux server. This performs a SYN-based port scan, a version scan, and the OS fingerprinting function, all with verbose output.

Figure 3.11 Full nmap Scan

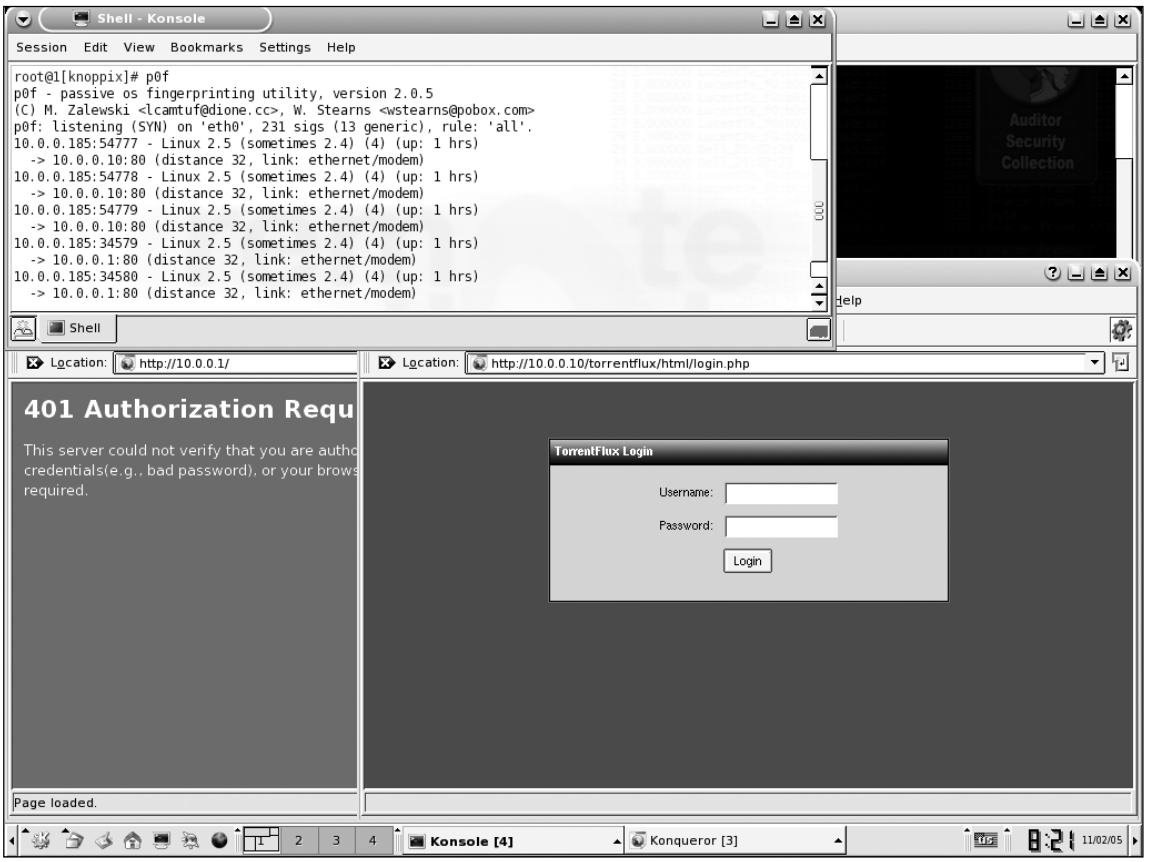


The version scanner picked up the version and protocol of OpenSSH, in use, the Web server name and version (Apache 2.0.53), and the Samba server version (3.*x*) and workgroup (HOMELAN). Also note that ftp, netbios-ns, ipp, and Nessus show closed, but present. In this case, the firewall rules are open for those services, but they are not currently running. Information like this would help you classify the system as a general server, and the open firewall ports could be scanned later to determine if the backend servers are running.

p0f: Passive OS Fingerprinting

p0f is the only passive fingerprinting tool included in the Auditor distribution. If you want to be extremely stealthy in your initial scan and enumeration processes, and don't mind getting high-level results for OS fingerprinting, p0f is the tool for you. It works by analyzing the responses from your target on innocuous queries, such as Web traffic, ping replies, or normal operations. p0f gives the best estimation on an operating system based on those replies, so it may not be as precise as other active tools, but it can still give you a good starting point. In Figure 3.12, p0f is used to check the version of a Fedora Linux server, and a Linksys Wireless router, both of which are returned as Linux. In this example, p0f is run in the background with an open Web browser to a Linksys access point's Web administration page, as well as a TorrentFlux (PHP-based BitTorrent client) login page. Both were detected as Linux, only correct in one case, as the Linksys AP is not a model based on Linux.

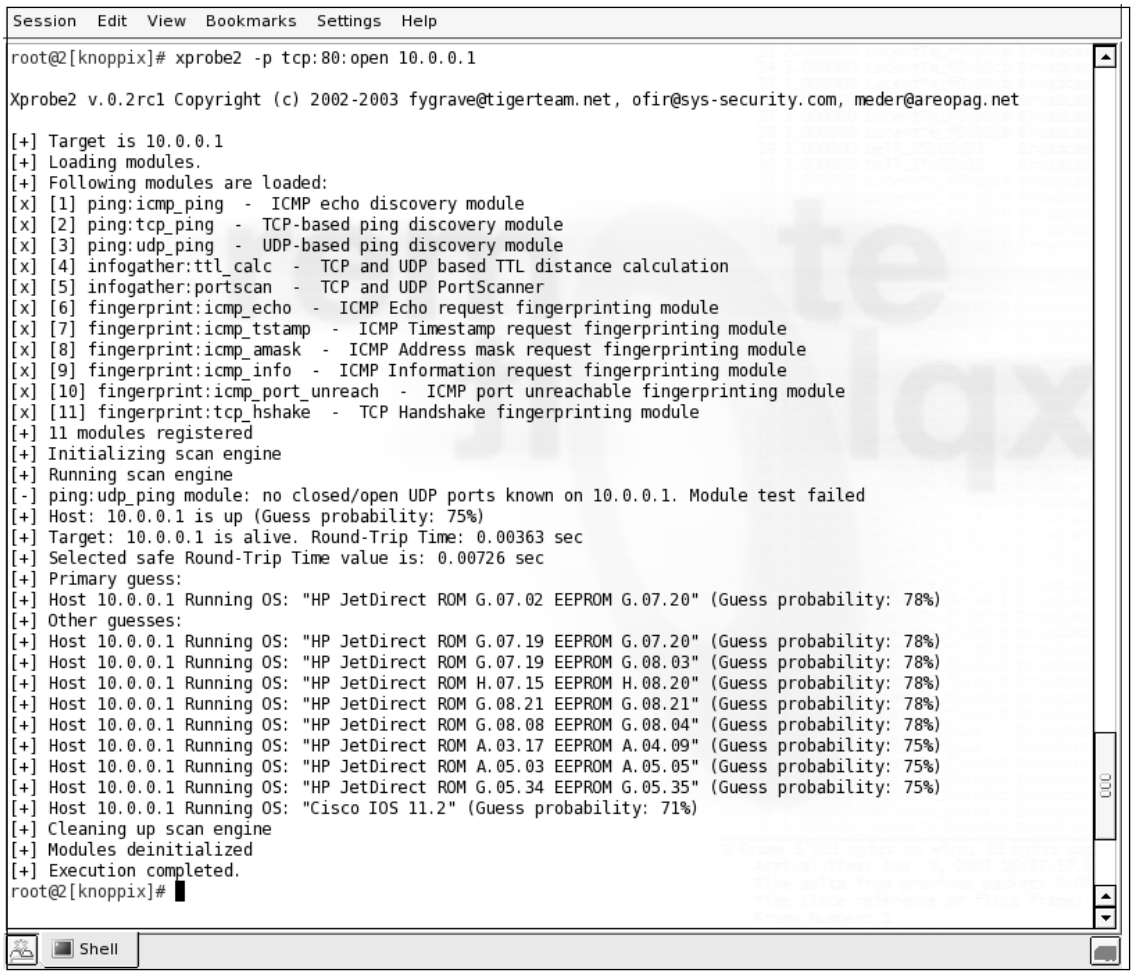
Figure 3.12 p0f OS Checking



Xprobe2: OS Fingerprinting

Xprobe2 is primarily an OS fingerprinter, but also has some basic port-scanning functionality built in to identify open or closed ports. You can also specify known open or closed ports, to which Xprobe2 performs several different TCP-, UDP-, and ICMP-based tests to determine the remote OS. The version supplied with Auditor is one version behind, but newer versions have more fingerprints. You will likely want to provide Xprobe2 with a known open or closed port for it to determine the remote OS, as in Figure 3.13.

Figure 2.13 Xprobe2 Fingerprinting



httprint

Suppose you run across a Web server and want to know the HTTP daemon running, without loading up a big fingerprinting tool that might trip IDS sensors? httprint is designed for just such a purpose. It only fingerprints http servers, and does both banner grabbing as well as signature matching against a provided signatures file. In Figure 3.14, we ran httprint against a Web server at 10.0.0.10, using *-h*, and designated the signatures with *-s* /opt/auditor/httprint/signatures.txt. The resulting banner specifies Apache 2.0.53 (Fedora), while the nearest signature match is Apache 2.0.*x*, which matches up. Beneath that output are all the signatures that were included, and then a score and confidence rating for that particular match.

Figure 3.14 httprint Web Server Fingerprint

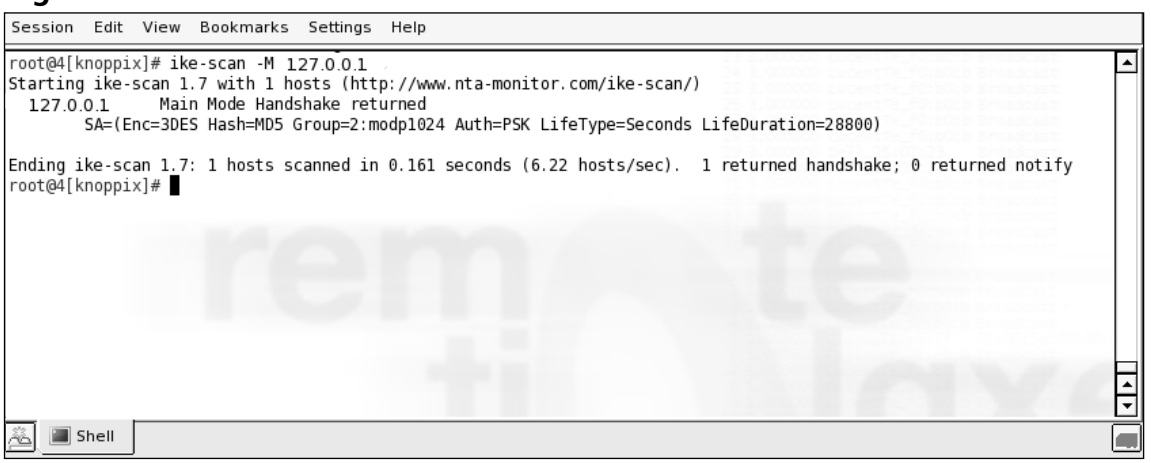


IKE-scan: VPN Assessment

One of the more common VPN implementations involves the use of IPsec tunnels. Different manufacturers have slightly different usages of IPsec, which can be discovered and fingerprinted using IKE-scan. IKE stands for Internet Key Exchange, and is used to provide a secure basis for establishing an IPsec-secured tunnel. IKE-scan can be run in two different modes, Main (*-M*) and Aggressive (*-A*), each of which can identify different VPN implementations. Both operate under the principle that VPN servers will attempt to establish communications to a client that only sends the initial portion of an IPsec handshake. An initial IKE packet is sent (with Aggressive mode, a UserID is also specified), and based on the time elapsed and types of responses sent, the VPN server can be identified based on service fingerprints. In addition to the VPN fingerprinting functionality, IKE-scan includes psk-crack,

which is a program used to dictionary crack pre-shared keys (psk) used for VPN logins. IKE-scan does not have fingerprints for all VPN vendors, and since the fingerprints change based on version increase, you may not find a fingerprint for your specific VPN, but you can still gain useful information, such as the Authentication type and encryption algorithm used, as shown in Figure 3.15

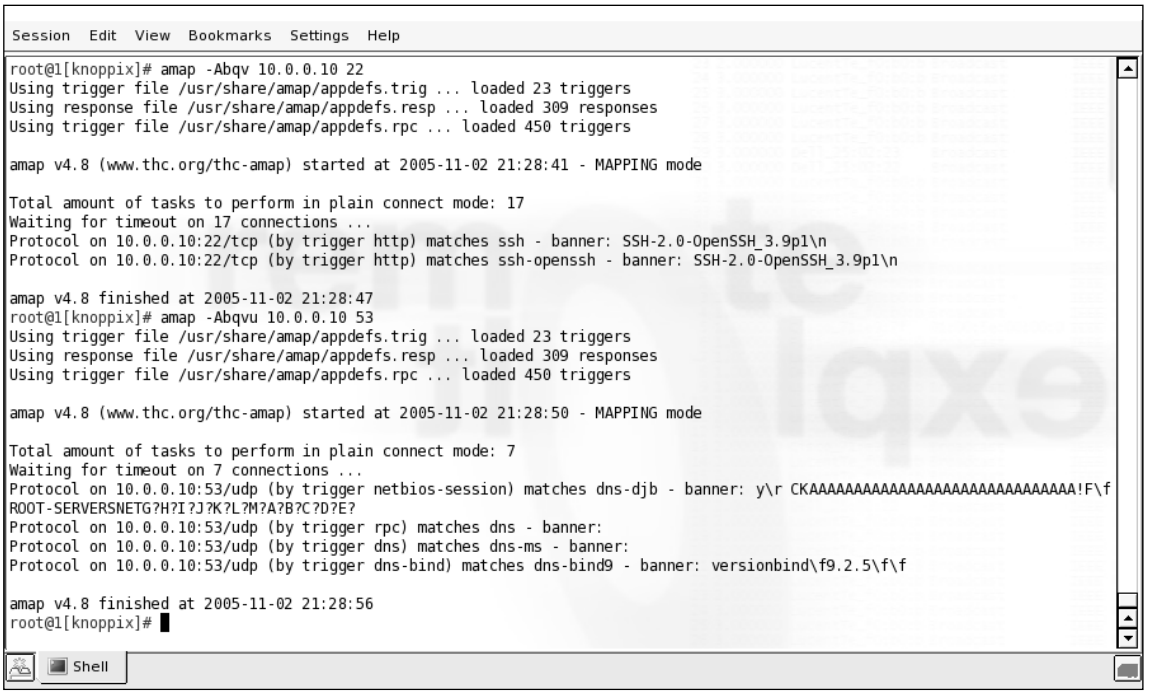
Figure 3.15 IKE-scan



amap: Application Version Detection

Sometimes, you may encounter a service that may not be easily recognizable by port number or immediate response. amap will send multiple queries and probes to a specific service, and then analyze the results, including returned banners, to identify what application or service is actually running on a specific port. There are options that allow you to minimize parallel attempts, or really stress the system with a large number of attempts, which may provide different information. You can also query a service once, and report back on the first matching banner reported, using the *-1* option. In Figure 3.16, amap is used to discover an OpenSSH server and a DNS server. The options used for these scans are to invoke mapping (*-A*), print any ASCII banner received (*-b*), do not mark closed and nonresponsive ports as identified or reported (*-q*), and use UDP ports (*-u*).

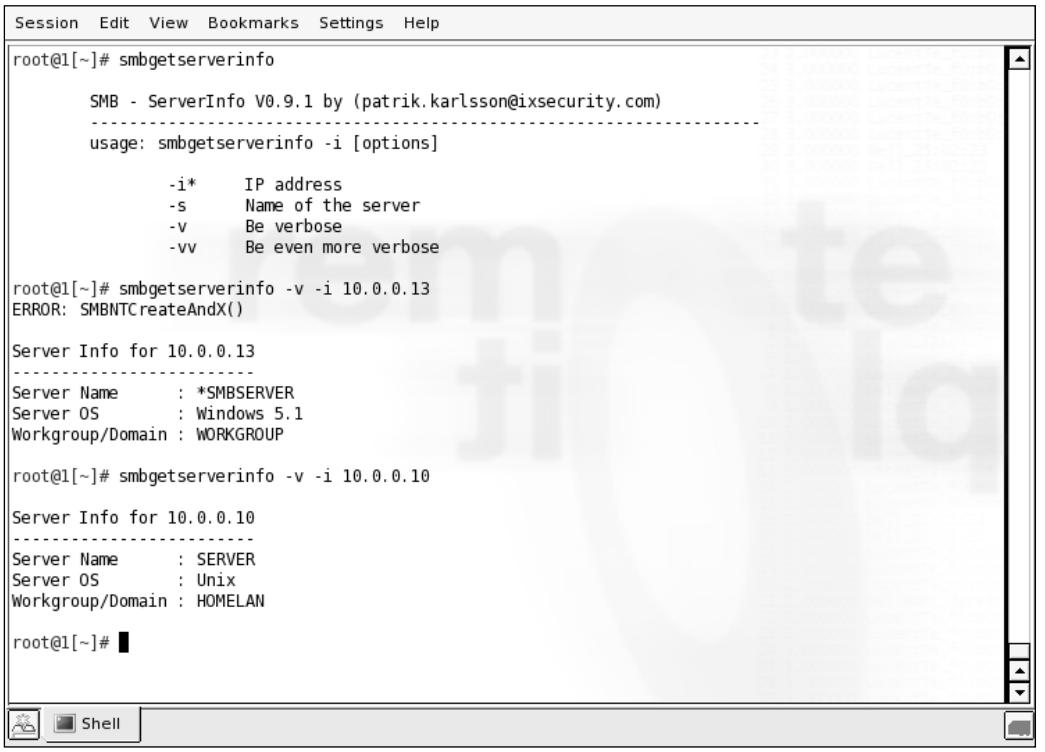
Figure 3.16 amap Detection Example



Windows Enumeration: smbgetserverinfo/smbdumpusers

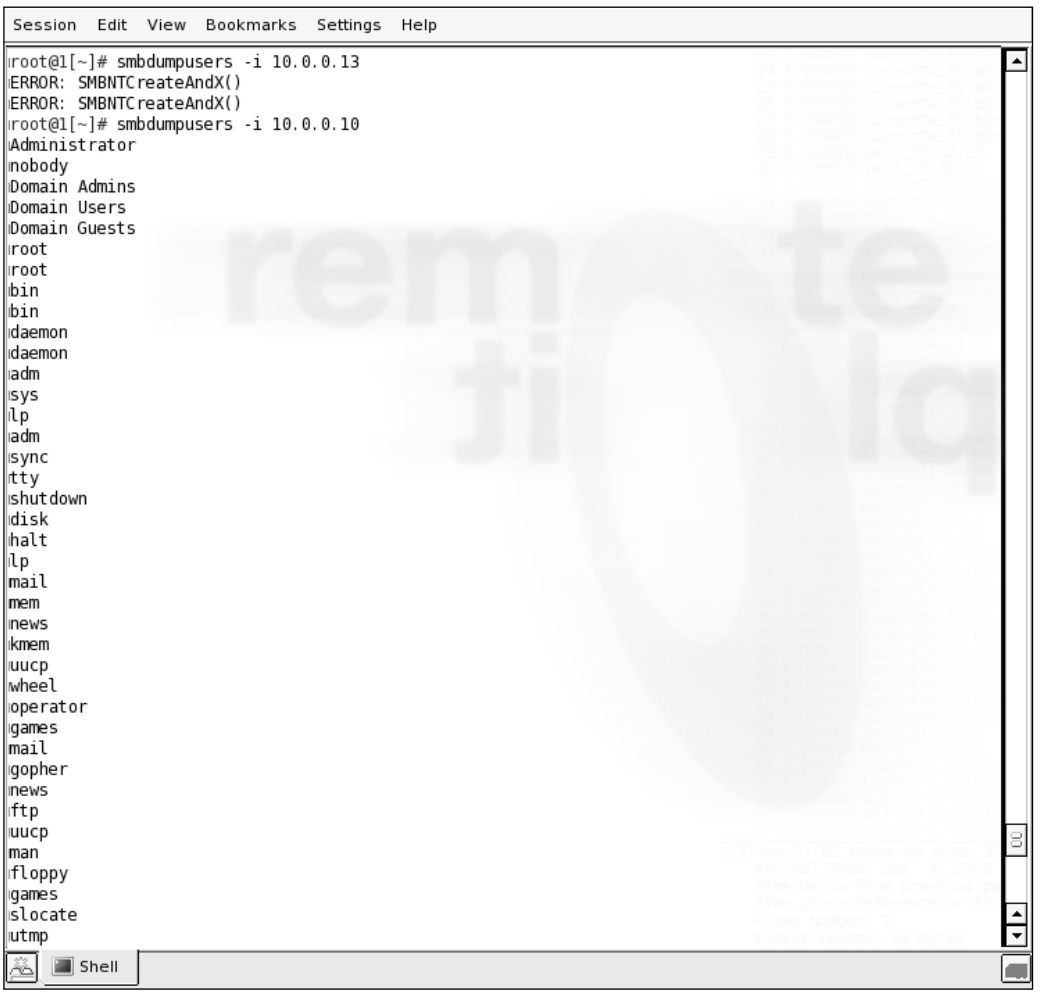
If TCP ports 135, 137, 139, or 445 are open, this indicates that the target machine is Windows-based or is most likely running a Windows-like service such as Samba. If you find these ports open, you should try to enumerate the system name and users via these services. In Windows, if the registry keys *RestrictAnonymous* and *RestrictAnonymousSAM* are set to 0, an anonymous user can connect to the system with a null session and dump the list of local user accounts and shared folders for the system. The *smb** tools shipped with Auditor do an excellent job of enumerating these services. However, these tools work much better against Windows 2000 and earlier versions, since Windows XP significantly locks down null sessions. Figure 3.17 shows the type of information returned from *smbgetserverinfo* on a Windows XP machine (10.0.0.13) and a Fedora Core 3 Linux server running Samba (10.0.0.10).

Figure 3.17 smbgetserverinfo Example



By connecting to a Samba server via a null session, you can get the Samba system name and the operating system version. The smbdumpusers program reveals much more information as shown in Figure 3.18. Although the Windows XP target does not return any information, the Linux target returns the listing of all local users, although the local Samba account of aaron is not displayed. Although these tools might be useful for older environments, when attacking newer Windows environments, other tools such as *nbtscan* and *Nessus* should be used instead.

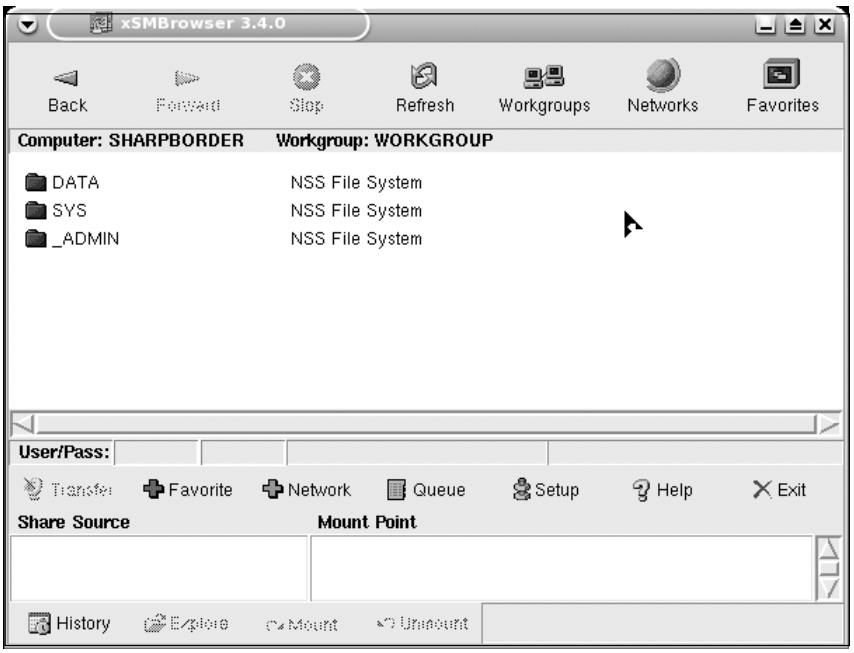
Figure 3.18 smbdumpusers Example



xSMBrowser

If SMB has been found for a target, you will need to try to find out more information about the offered shares. xSMBrowser offers a GUI environment for connecting and enumerating a system within either a workgroup or a domain. Launch xSMBrowser, and select **Samba Config** to start the browsing. It will execute nmblookup to search the workgroup/domain, and any systems returned will be run through smbclient to attempt to list and connect to remote shares. Depending on the security settings of the target system, you will be able to view and browse the shares, as demonstrated in Figure 3.19.

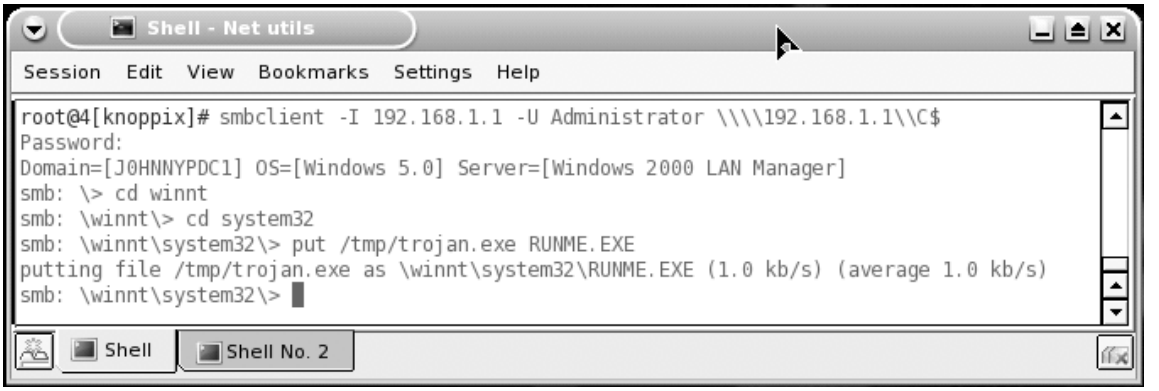
Figure 3.19 xSMBrowser Lookup and Enumeration



smbclient

If you are enumerating a target, and determine that there is an accessible file share, how can you access it for the purpose of retrieving or sending files? One of the oldest methods is the use of smbclient, which is a file transfer client for SMB networks that operates like an FTP client. You can specify the username, using the *-U* parameter and target host using *-I*. Figure 3.20 shows a successful connect to a target system using smbclient for the purpose of sending a file that may be later used in the penetration test.

Figure 3.20 smbclient Sending File



Summary

Ok, we have covered many tools in this chapter, and definitely many different switches/flags/parameters for those tools. Table 3.2 lists all the tools mentioned here, a summary of the flags used, and their expected purpose.

Table 3.2 Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
tee	-a filename	Append to filename, no switch overwrites file
nmap	-sP	Ping sweep using both ICMP and TCP ACK (as root)
	-P0	Do not ICMP ping target before scanning
	-PS	Use TCP ACK probe as ping type
	-PP	Use ICMP Timestamp requests as probe
	-PM	Use ICMP Netmask request as probe
	-oA	Output all types of log files, standard text, XML, and greppable
	—resume outputfile.gnmap	Resume a previously cancelled nmap scan
	-sT	OS connect() based scan, default with user privileges
	-sS	SYN scan, default with root privileges
	-b user:pass @server:ftpport	FTP bounce scan from server

Continued

Table 3.2 continued Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
	-sI zombiehost:port	Idlescan host through zombiehost on port
	-O	Use OS fingerprinting methods
	-iL hostfile	Launch using hostfile as target list
	-T[0,1,2,3,4,5]	Set scan timing from Paranoid (T0) to Insane (T5)
	-p[ports/portlist /port1-port2]	Scan designated ports only
	-sU	Launch UDP port scan
	-sV	Initiate service scan on detected ports
	-sA	Perform both service scan (-sV) and OS fingerprint (-O)
	-v	Enable verbose output
netenum	<timeout>	Specify timeout for moving to next port; without timeout specified, netenum will expand given port list to stdout and exit
unicornscan	target:q	Scan target range using "Quick" ports
scanrand	-b bandwidth [b/k/m/g]	Scan target using bandwidth specified in bytes, kilobytes, megabytes, gigabytes per second
	-e	Show target host, even if ports are non-responsive
	-v	Show sent and received packets
	-p number	Set source TCP port to number
	target:port.ports, port-range	Scan target on specified ports
p0f	-i interface	Launch p0f on specified interface
xprobe2	-p tcp/udp:port:status	Launch xprobe2 against either TCP or UDP port with a status of open or closed
httprint	-h	Specify target hostname or IP address
	-s /path/to/ signatures.txt	Specify signatures.txt file used for scanning (/opt/auditor/httprint/signatures.txt is default for Auditor)
ike-scan	-M	Use Main mode (default)
	-A	Use Aggressive mode where a UserID is specified

Continued

Table 3.2 continued Tools, Switches, and Purpose

Tool	Switches	Intended Purpose
amap	-A target port	Map (fingerprint) services found on target for port
	-b	Print ASCII banners received
	-q	Do not mark or report closed or nonresponsive ports
	-u target port	Use UDP port on target
smbgetserver info	-i address	Launch against IP address specified
	-v	Respond with verbose output
smbdumpusers	-i address	Launch against IP address specified
xsmbrowser	No parameters accepted	
smbclient	-I address	Launch against IP address specified
	-U username	Attempt to connect using username
nbtscan	-v	Respond with verbose output
	-f hostfile	Scan addresses listed in hostfile
smb-nat	-u list	Attempt to connect to target using usernames from list
	-p list	Attempt to connect to target using passwords from list

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Why do you use nmap, amap, and other tools when simply running Nessus can give you all of the same information in one pass?

A: Although Nessus is a great tool, it is just one tool. Sometimes, you will get different information returned about the same target using different tools, and that different information allows you to find out what is closer to the truth.

Q: You spend a lot of time double-checking and triple-checking the identity of a host before you go active and scan them; why take so much time when you could be actively gathering information?

A: Penetration testing should be performed with a well-defined scope. Only activities that you perform within that scope are authorized. Suppose you do not perform due diligence by checking out your target range and "accidentally" scan an IP neighbor's target. Even if you do not cause damage, you may be liable for simply performing the scan, much less if you penetrate the target. With that being a negative reason, you can also find "infringing" targets that are not related to your client, but may be purporting themselves to be. This can be as valuable as finding an open hole in their Web server.

Chapter 4

Introducing Intrusion Detection and Snort

Solutions in this chapter:

- How and IDS Works
- Where Snort Fits
- Snort System Requirements
- Exploring Snort's Features
- Using Snort on Your Network
- Snort and Your Network Architecture
- Pitfalls When Running Snort
- Security Considerations with Snort

- ☑ Summary
- ☑ Solutions Fast Track
- ☑ Frequently Asked Questions

Introduction

An IDS is the high-tech equivalent of a burglar alarm, one that is configured to monitor information gateways, hostile activities, and known intruders. An IDS is a specialized tool that knows how to parse and interpret network traffic and/or host activities. This data can range from network packet analysis to the contents of log files from routers, firewalls, and servers, local system logs and access calls, network flow data, and more. Furthermore, an IDS often stores a database of known attack signatures and can compare patterns of activity, traffic, or behavior it sees in the data it's monitoring against those signatures to recognize when a close match between a signature and current or recent behavior occurs. At that point, the IDS can issue alarms or alerts, take various kinds of automated actions ranging from shutting down Internet links or specific servers to launching back-traces, and make other active attempts to identify attackers and collect evidence of their nefarious activities.

By analogy, an IDS does for a network what an antivirus software package does for files that enter a system: it inspects the contents of network traffic to look for and deflect possible attacks, just as an antivirus software package inspects the contents of incoming files, e-mail attachments, active Web content, and so forth to look for *virus signatures* (patterns that match known malware) or for possible *malicious actions* (patterns of behavior that are at least suspicious, if not downright unacceptable).

To be more specific, intrusion detection means detecting unauthorized use of or attacks upon a system or network. An IDS is designed and used to detect such attacks or unauthorized use of systems, networks, and related resources, and then in many cases to deflect or deter them if possible. Like firewalls, IDSes can be software-based or can combine hardware and software in the form of preinstalled and preconfigured stand-alone IDS devices. IDS software may run on the same devices or servers where firewalls, proxies, or other boundary services operate, though separate IDS sensors and managers are more popular. Nevertheless, an IDS *not* running on the same device or server where the firewall or other services are installed will monitor those devices with particular closeness and care. Although such devices tend to be deployed at network peripheries, IDSes can detect and deal with insider attacks as well as external attacks, and are often very useful in detecting violations of corporate security policy and other internal threats.

You are likely to encounter several kinds of IDSes in the field. First, it is possible to distinguish IDSes by the kinds of activities, traffic, transactions, or systems they monitor. IDSes that monitor network links and backbones looking for attack signatures are called *network-based IDSes*, whereas those that operate on hosts and defend and monitor the operating and file systems for signs of intrusion and are called *host-based IDSes.* Groups of IDSes functioning as remote sensors and reporting to a central management station are known as distributed IDSes (DIDSes). A *gateway IDS* is a network IDS deployed at the gateway between your network and another network, monitoring the traffic passing in and out of your network at the transit point. IDSes that focus on understanding and parsing application-specific

traffic with regard to the flow of application logic as well as the underlying protocols are often called *application IDSes*.

In practice, most commercial environments use some combination of network-, host-, and/or application-based IDSes to observe what is happening on the network while also monitoring key hosts and applications more closely. IDSes can also be distinguished by their differing approaches to event analysis. Some IDSes primarily use a technique called *signature detection*. This resembles the way many antivirus programs use virus signatures to recognize and block infected files, programs, or active Web content from entering a computer system, except that it uses a database of traffic or activity patterns related to known attacks, called *attack signatures*. Indeed, signature detection is the most widely used approach in commercial IDS technology today. Another approach is called *anomaly detection*. It uses rules or predefined concepts about "normal" and "abnormal" system activity (called *heuristics*) to distinguish anomalies from normal system behavior and to monitor, report, or block anomalies as they occur. Some anomaly detection IDSes implement user profiles. These profiles are baselines of normal activity and can be constructed using statistical sampling, rule-base approaches, or neural networks.

Hundreds of vendors offer various forms of commercial IDS implementations. Most effective solutions combine network- and host-based IDS implementations. Likewise, the majority of implementations are primarily signature-based, with only limited anomaly-based detection capabilities present in certain specific products or solutions. Finally, most modern IDSes include some limited automatic response capabilities, but these usually concentrate on automated traffic filtering, blocking, or disconnects as a last resort. Although some systems claim to be able to launch counterstrikes against attacks, best practices indicate that automated identification and back-trace facilities are the most useful aspects that such facilities provide and are therefore those most likely to be used.

IDSes are classified by their functionality and are loosely grouped into the following three main categories:

- Network-based intrusion detection system (NIDS)
- Host-based intrusion detection system (HIDS)
- Distributed intrusion detection system (DIDS)

How an IDS Works

We've already touched on this to some degree in our survey of the different kinds of IDSes out there, but let's take a look at exactly what makes an IDS tick. First, you have to understand what the IDS is watching. The particular kinds of data input will depend on the kind of IDS (indeed, what sorts of information an IDS watches is one of the hallmarks used to classify it), but in general there are three major divisions:

- Application-specific information such as correct application data flow
- Host-specific information such as system calls used, local log content, and file system permissions
- Network-specific information such as the contents of packets on the wire or hosts known to be attackers

A DIDS may watch any or all of these, depending on what kinds of IDSes its remote sensors are. The IDS can use a variety of techniques in order to gather this data, including packet sniffing (generally in promiscuous mode to capture as much network data as possible), log parsing for local system and application logs, system call watching in the kernel to regulate the acceptable behavior of local applications, and file system watching in order to detect attempted violation of permissions.

After the IDS has gathered the data, it uses several techniques to find intrusions and intrusion attempts. Much like firewalls, an IDS can adopt a known-good or a known-bad policy. With the former technique, the IDS is set to recognize good or allowed data, and to alert on anything else. Many of the anomaly detection engines embrace this model, triggering alerts when anything outside of a defined set of statistical parameters occurs. Some complex protocol models also operate on known-good policies, defining the kinds of traffic that the protocol allows and alerting on anything that breaks that mold. Language-based models for application logic also tend to be structured as known-good policies, alerting on anything not permitted in the predefined structure of acceptable language or application flow.

Known-bad policies are much simpler, as they do not require a comprehensive model of allowed input, and alert only on data or traffic known to be a problem. Most signature-based IDS engines work from a known-bad model, with an ever-expanding database of malicious attack signatures. Known-good and known-bad policies can work in conjunction within a single IDS deployment, using the known-bad signature detection and the known-good protocol anomaly detection in order to find more attacks.

Finally, we should consider what the IDS does when it finds an attempted attack. There are two general categories of response: passive response, which may generate alerts or log entries but does not interfere with or manipulate the network traffic, and active response, which may send reset packets to disrupt Transmission Control Protocol (TCP) connections, drop traffic if the IDS is inline, add the attacking host to block lists, or otherwise actively interact with the flow of dubious activity.

Having outlined these principles in the abstract, let's take a look at some real network-based attacks.

What Will an IDS Do for Me?

The strengths of IDSes are their capability to continuously watch packets on your network, understand them in binary, and alert you when something suspicious that matches a signa-

ture occurs. Unlike human security analysts, the speed of IDS detection allows alerting and response almost immediately, even if it's 3 A.M. and everyone's sleeping. (The alerting capability of IDSes can allow you to page people and wake them up, or, if you're deploying an IDS in inline mode or an intrusion prevention system [IPS], block the suspicious traffic, and potentially other traffic from the attacking host.) An IDS can allow you to read gigabytes of logs daily, looking for specific issues and violations. The potential enhancement of computing and analysis power is tremendous, and a well-tuned IDS will act as a force multiplier for a competent system/network administrator or security person, allowing them to monitor more data from more systems. By letting you know quickly when it looks like you are under attack, potential compromises may be prevented or minimized.

It is important to realize that any IDS is likely to create tremendous amounts of data no matter how well you tune it. Without tuning, most IDSes will create so much data and so many false positives that the analysis time may swamp response to the legitimate alerts in a sea of false alerts. A new IDS is almost like a new baby—it needs lots of care and feeding to be able to mature in a productive and healthy way. If you don't tune your IDS, you might as well not have it.

Another positive feature of an IDS is that it will allow the skilled analyst to find subtle trends in large amounts of data that might not otherwise be noticed. By allowing correlation between alerts and hosts, the IDS can show patterns that might not have been noticed through other means of network analysis. This is one example of how an IDS can supplement your other network defenses, working cooperatively to enact a defense-in-depth strategy.

What Won't an IDS Do for Me?

No IDS will replace the need for staffers knowledgeable about security. You'll need skilled analysts to go through those alerts that the IDS produces, determining which are real threats and which are false positives. Although the IDS can gather data from many devices on a network segment, they still won't understand the ramifications of threats to each machine, or the importance of every server on the network. You need clever, savvy people to take action on the information that the IDS provides.

In addition, no IDS will catch every single attack that occurs, or prevent people from trying to attack you. The limitations of any kind of IDS and the timing between the development of new attacks and the development of signatures or the ability to hide within acceptable parameters of an anomaly-based system make it exceedingly likely that there will be a small window in which 0-day attacks will not be detected by a given IDS. The Internet can be a cruel and hostile place, and although it's advisable to implement strong network defenses and prepare to be attacked, IDSes cannot psychically make people decide not to attack your network after all. In most cases, an IDS will not prevent attacks from succeeding automatically, as its function is primarily to detect and alert. There are some mechanisms that

do address this problem—inline IDS, or IPS, for example—but in most cases, an IDS will not automatically defeat attacks for you. This is one of the reasons that an IDS should be seen as a complement to your other network defenses such as firewalls, antivirus software, and the like, rather than as a replacement for them.

Where Snort Fits

Snort is an open source network IDS capable of performing real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort can perform protocol analysis and content searching/matching, and you can use it to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, Common Gateway Interface (CGI) attacks, Server Message Block (SMB) probes, operating system fingerprinting attempts, and much more. Snort is rapidly becoming the tool of choice for intrusion detection.

You can configure Snort in three main modes: sniffer, packet logger, and network intrusion detection. Sniffer mode simply reads the packets off the network and displays them in a continuous stream on the console. Packet logger mode logs the packets to the disk. Network intrusion detection mode is the most complex and configurable, allowing Snort to analyze network traffic for matches against a user-defined ruleset and to perform one of several actions, based on what it sees.

In addition to the community signatures provided with Snort and the Sourcefire VDB signatures available for download to registered users, you can write your own signatures with Snort to suit the particular needs of your network. This capability adds immense customization and flexibility to the Snort engine, allowing you to suit the unique security needs of your own network. In addition, there are several online communities where leading-edge intrusion analysts and incident responders swap their newest Snort rules for detecting fresh exploits and recent viruses.

Snort's network pattern matching behavior has several immediately practical applications. For example, it allows the detection of hosts infected with viruses or worms that have distinctive network behavior. Because many modern worms spread by scanning the Internet and attacking hosts they deem vulnerable, signatures can be written either for this scanning behavior or for the exploit attempt itself. Although it is not the job of the IDS to clean up infected machines, it can help identify infected machines. In cases of massive virus infection, this identification capability can be immensely useful. In addition, watching for the same behavior after supposed virus cleanup can help to confirm that the cleanup was successful.

Snort also has signatures that match the network behavior of known network reconnaissance and exploit tools. Although for the most part, rule writers make an effort to match the signature of the exploit and not of a particular tool, sometimes it's helpful to be able to identify the tool scanning or attacking you. For example, there are rules that identify the SolarWinds scanner's tendency to embed its name in the payload of its scanning Internet Control Message Protocol (ICMP) packets, making for easy device identification. The vast

majority of exploits that end up in popular tools such as Metasploit have signatures in the Snort rulebases, making them detectable by their network behavior.

Snort System Requirements

Before getting a system together, you need to know a few things. First, Snort data can take up a lot of disk space, and, second, you'll need to be able to monitor the system remotely. The Snort system we maintain is in our machine room (which is cold, and a hike downstairs).

Because we're lazy and don't want to hike downstairs, we would like to be able to maintain it remotely *and* securely. For Linux and UNIX systems, this means including Secure Shell (SSH) and Apache with Secure Sockets Layer (SSL). For Windows, this would mean Terminal Services (with limitation on which users and machines can connect and Internet Information Servers [IIS]).

Hardware

It's difficult to give hard-and-fast requirements on what you'll need to run Snort because the hardware requirements are tremendously variable depending on the amount of traffic on your network and how much of that you're trying to process and store. Busy enterprise networks with thousands of active servers are going to have much greater requirements than a poky home network with one client machine on it. However, we can provide general guidelines.

At a bare minimum level, Snort does not have any particular hardware requirements that your OS doesn't already require to run. Running any application with a faster processor usually makes the application work faster. However, your network connection and hard drive will limit the amount of data you can collect.

One of the most important things you'll need, especially if you're running Snort in NIDS mode, is a really big, reasonably fast hard drive. If you're storing your data as either syslog files or in a database, you'll need a lot of space to store all the data that the Snort's detection engine uses to check for rule violations. If your hard drive is too small, there is a good chance that you will be unable to write alerts to either your database or log files. For example, our current setup for a single high-traffic enterprise Snort sensor is a 100GB partition for /var (for those of you not familiar with Linux/UNIX systems, /var is where logs, including Snort data, are most likely to be stored). Some high-end deployments even use RAID arrays for storage.

You will need to have a network interface card (NIC) as fast or faster than the rest of your network to collect all the packets on your network. For example, if you are on a 100MB network, you will need a 100MB NIC to collect the correct amount of packets. Otherwise, you will miss packets and be unable to accurately collect alerts. A highly recommended hardware component for Snort is a second Ethernet interface. One of the interfaces is necessary for typical network connectivity (SSH, Web services, and so forth), and the other

interface is for Snorting. This sensing interface that does the "snorting" is your "Snort sensor." Having separate interfaces for sensor management and for network sniffing enhances security because it allows you to strongly restrict which machines are able to access the management interface without interfering with your promiscuous packet sniffing on the "snorting" interface.

Given the new improvements to the Snort engine, we also suggest not shorting your system on memory. Since Snort has a bigger memory footprint than earlier versions, it's useful to make sure that your sensors have enough RAM to handle the volume of traffic that you're getting. If you notice performance lag, it's worthwhile to make sure that your system is not swapping memory intensively.

Operating System

Snort was designed to be a lightweight network intrusion system. Currently, Snort can run on x86 systems Linux, FreeBSD, NetBSD, OpenBSD, and Windows. Other systems supported include Sparc Solaris, x86 Mac OS X, PowerPC Mac OS X and MkLinux, and PA-RISC HP-UX. In short, Snort will run on just about any modern OS. For this book, we'll obviously be focusing on Linux.

There is an ongoing argument regarding the best OS on which to run Snort. A while back, the *BSDs had the better IP stack, but since Linux has gone to the 2.4 kernel, the IP stacks are comparable.

Other Software

Once you have the basic OS installed, you're ready to go. Make sure that you have the prerequisites before you install:

- autoconf and automake*
- gcc*
- lex and yacc (or the GNU implementations flex and bison, respectively)
- the latest libcap from tcpdump.org

NOTE

These are only necessary if you're compiling from source code. If you are using Linux RPMs or Debian packages or a Windows port installer, you do not need these. AND YOU SHOULD NOT HAVE THEM ON A PRODUCTION IDS SENSOR! Once you have compiled and put Snort into place, all of the tools for compiling it should be removed from any sensor that you expect to put into your production environment.

You can also install the following optional software products:

- MySQL, Postgres, or Oracle (SQL databases)
- smbclient if using WinPopup messages
- Apache or another Web server
- PHP or Perl, if you have plug-ins that require them
- SSH for remote access (or Terminal Server with Windows)
- Apache with SSL capabilities for monitoring (or IIS for Windows)

There's more detail on installation in Chapter 5, "Installing Snort."

Exploring Snort's Features

In the Introduction, we provided you with a brief overview of Snort's most important features that make it very powerful: packet sniffing, packet logging, and intrusion detection. Before learning the details of Snort's features, you should understand Snort's architecture. Snort has several important components such as preprocessors and alert plug-ins, most of which can be further customized with plug-ins for your particular Snort implementation. These components enable Snort to manipulate a packet to make the contents more manageable by the detection engine and the alert system. Once the packet has been passed through the preprocessors, passed through the detection engine, and then sent through the alert system, it can be handled by whatever plug-ins you have chosen to handle alerting. It sounds complicated initially, but once you understand the architecture, Snort makes a lot more sense.

Snort's architecture consists of four basic components:

- The sniffer
- The preprocessor
- The detection engine
- The output

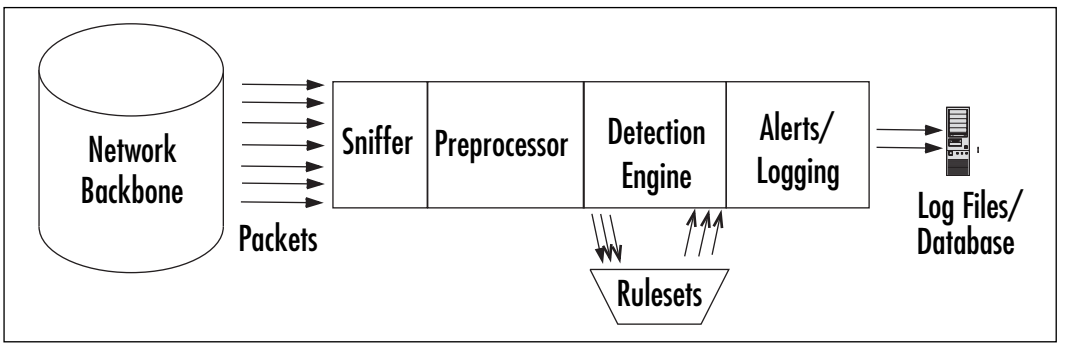
In its most basic form, Snort is a packet sniffer. However, it is designed to take packets and process them through the preprocessor, and then check those packets against a series of rules (through the detection engine).

Figure 4.1 offers a high-level view of the Snort architecture. In its simplest form, Snort's architecture is similar to a mechanical coin sorter.

1. It takes all the coins (packets from the network backbone).
2. Then it sends them through a chute to determine if they are coins and how they should roll (the preprocessor).

3. Next, it sorts the coins according to the coin type. This is for storage of quarters, nickels, dimes, and pennies (on the IDS this is the detection engine).
4. Finally, it is the administrator's task to decide what to do with the coins—usually you'll roll them and store them (logging and database storage).

Figure 4.1 Snort Architecture



The preprocessor, the detection engine, and the alert components of Snort are all plug-ins. Plug-ins are programs that are written to conform to Snort's plug-in API. These programs used to be part of the core Snort code, but they were separated to make modifications to the core source code more reliable and easier to accomplish.

Packet Sniffer

A packet sniffer is a device (either hardware or software) used to tap into networks. It works in a similar fashion to a telephone wiretap, but it's used for data networks instead of voice networks. A network sniffer allows an application or a hardware device to eavesdrop on data network traffic. In the case of the Internet, this usually consists of IP traffic, but in local LANs and legacy networks, it can be other protocol suites, such as IPX and AppleTalk traffic.

Because IP traffic consists of many different higher-level protocols (including TCP, UDP, ICMP, routing protocols, and IPSec), many sniffers analyze the various network protocols to interpret the packets into something human-readable.

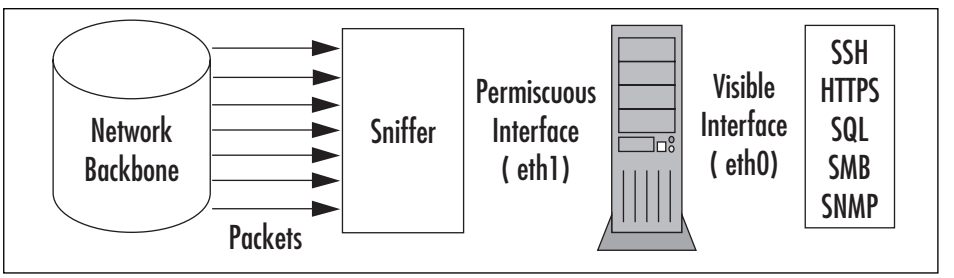
Packet sniffers have various uses:

- Network analysis and troubleshooting
- Performance analysis and benchmarking
- Eavesdropping for clear-text passwords and other interesting tidbits of data

Encrypting your network traffic can prevent people from being able to sniff your packets into something readable. Like any network tool, packet sniffers can be used for good and evil.

As Marty Roesch said, he named the application because it does more than sniffing—it snorts. The sniffer needs to be set up to obtain as many packets as possible. As a sniffer, Snort can save the packets to be processed and viewed later as a packet logger. Figure 4.2 illustrates Snort's packet-sniffing ability.

Figure 4.2 Snort's Packet-Sniffing Functionality



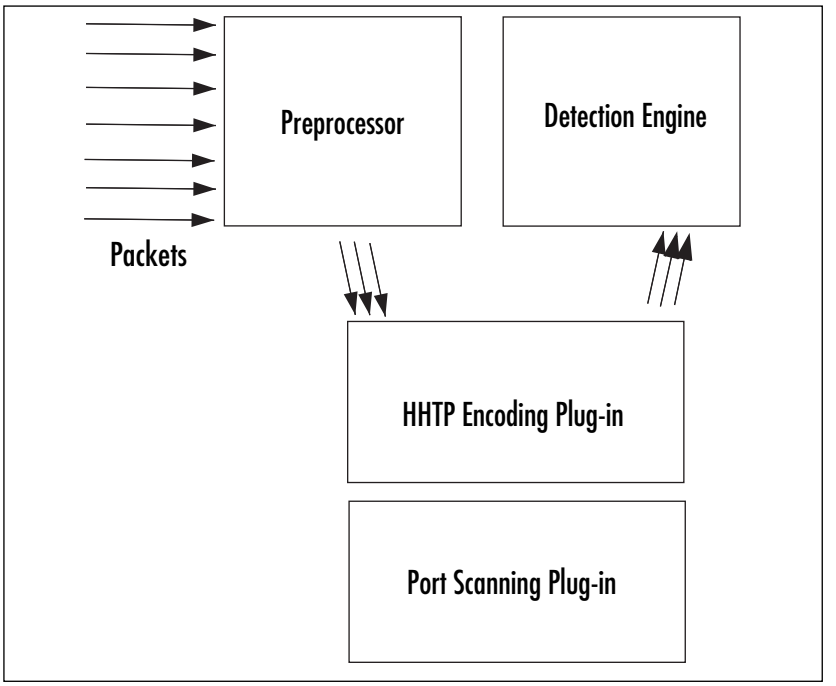
Preprocessor

At this point, our coin sorter has obtained all the coins it can (packets from the network) and is ready to send the packets through the chute. Before rolling the coins (the detection engine), the coin sorter needs to determine if they are coins, and if so, what sorts.

This is done through the preprocessors. A preprocessor takes the raw packets and checks them against certain plug-ins (like an RPC plug-in, an HTTP plug-in, and a port scanner plug-in). These plug-ins check for a certain type of behavior from the packet. Once the packet is determined to have a particular type of "behavior," it is then sent to the detection engine. From Figure 4.3, you can see how the preprocessor uses its plug-ins to check a packet. Snort supports many kinds of preprocessors and their attendant plug-ins, covering many commonly used protocols as well as larger-view protocol issues such as IP fragmentation handling, port scanning and flow control, and deep inspection of richly featured protocols (such as the HTTPinspect preprocessor handles).

This is an incredibly useful feature for an IDS because plug-ins can be enabled and disabled as they are needed at the preprocessor level, allocating computational resources and generating alerts at the level optimal for your network. For example, say that you're fed up with the constant rate of port scans of your network, and you don't want to see those alerts any more. In fact, you never want to hear about a port scan again. If that's the case, you can say you don't care about port scans coming into your network from the outside world and disable that plug-in while still continuing to use the others to examine other network threats. It's a modular configuration, rather than an all-or-nothing scenario.

Figure 4.3 Snort's Preprocessor



Detection Engine

Once packets have been handled by all enabled preprocessors, they are handed off to the detection engine. The detection engine is the meat of the signature-based IDS in Snort. The detection engine takes the data that comes from the preprocessor and its plug-ins, and that data is checked through a set of rules. If the rules match the data in the packet, they are sent to the alert processor.

Earlier in this chapter, we described Snort as a signature-based IDS. The signature-based IDS function is accomplished by using various rulesets. The rulesets are grouped by category (Trojan horses, buffer overflows, access to various applications) and are updated regularly.

The rules themselves consist of two parts:

- **The rule header** The rule header is basically the action to take (log or alert), type of network packet (TCP, UDP, ICMP, and so forth), source and destination IP addresses, and ports
- **The rule option** The option is the content in the packet that should make the packet match the rule.

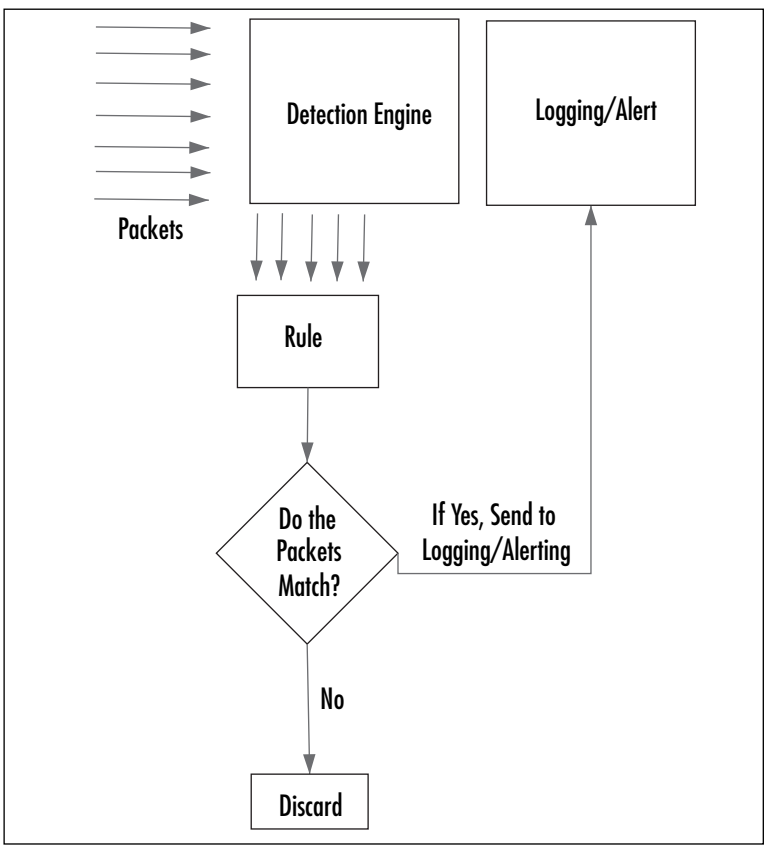
The detection engine and its rules are the largest portion (and steepest learning curve) of new information to learn and understand with Snort. Snort has a particular syntax that it

uses with its rules. Rule syntax can involve the type of protocol, the content, the length, the header, and other various elements, including garbage characters for defining butter overflow rules.

Once you get it working and learn how to write Snort rules, you can fine-tune and customize Snort's IDS functionality. You can define rules that are particular to your environment and customize however you want.

The detection engine is the part of the coin sorter that actually rolls the coins based on the type. The most common American coins are the quarter, dime, nickel, and penny. However, you might get a coin that doesn't match, like the Kennedy half-dollar, and discard it. This is illustrated in Figure 4.4.

Figure 4.4 Snort's Detection Engine



Alerting/Logging Component

After the Snort data goes through the detection engine, it needs to go out somewhere. If the data matches a rule in the detection engine, an alert is triggered. Alerts can be sent to a log

file, through a network connection, through UNIX sockets or Windows Popup (SMB), or SNMP traps. The alerts can also be stored in an SQL database such as MySQL and Postgres.

You can also use additional tools with Snort, including various plug-ins for Perl, PHP, and Web servers to display the logs through a Web interface. Logs are stored in either text files (by default in /var/log/snort) or in a database such as MySQL and Postgres.

Like the detection engine and the preprocessor, the alert component uses plug-ins to send the alerts to databases and through networking protocols such as SNMP traps and WinPopup messages. See Figure 4.5 for an illustration of how this works.

Additionally, with syslog tools such as Swatch, Snort alerts can be sent via e-mail to notify a system administrator in real time so no one has to monitor the Snort output all day and night.

Table 4.1 lists a few examples of various useful third-party programs and tools.

Table 4.1 Useful Snort Add-Ons

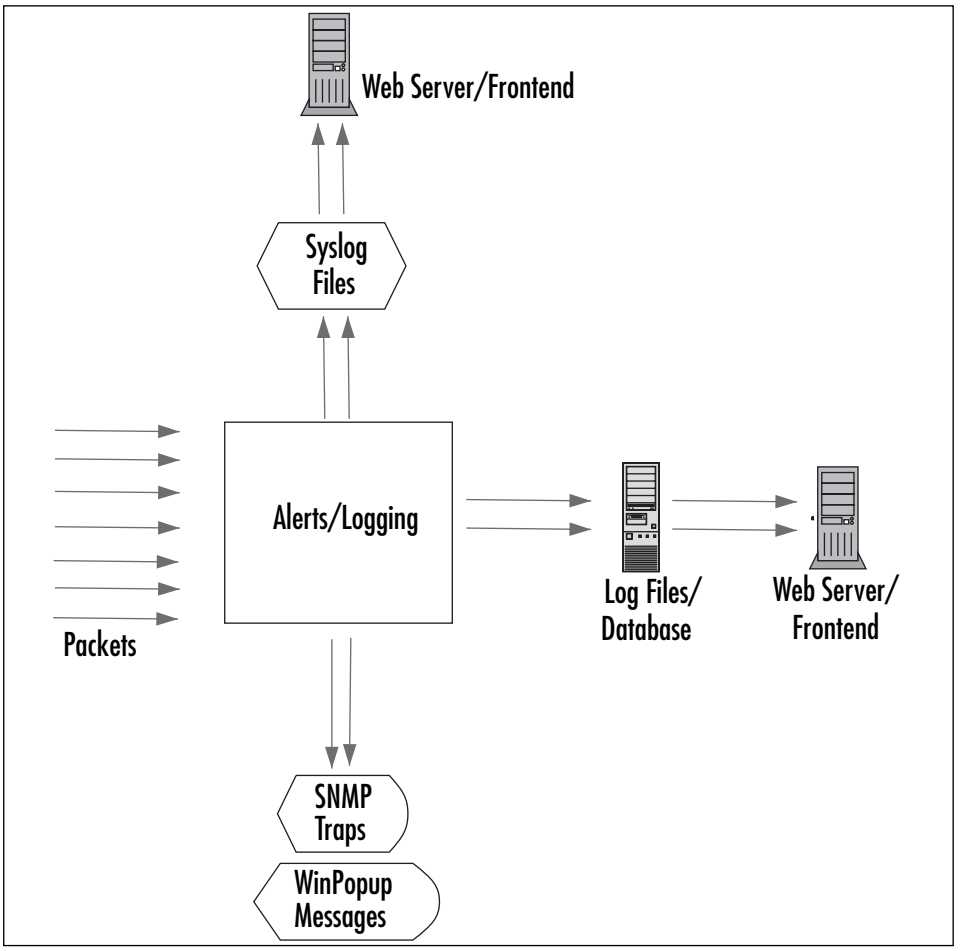
Output Viewer	URL	Description
SnortSnarf	www.silicondefense.com/software/snortsnarf	A Snort analyzer by Silicon Defense used for diagnostics. The output is in HTML.
Snortplot.php	www.snort.org/dl/contrib/data_analysis/snortplot.pl	A Perl script that will graphically plot your attacks.
Swatch	http://swatch.sourceforge.net	A real-time syslog monitor that also provides real-time alerts via e-mail.
ACID	http://acidlab.sourceforge.net	The Analysis Console for Intrusion Databases. Provides logging analysis for Snort. Requires PHP, Apache, and the Snort database plug-in. Since this information is usually sensitive, it is strongly recommended that you encrypt this information by using mod_ssl with Apache or Apache-SSL. ACID is basically deprecated and not being developed

Continued

Table 4.1 continued Useful Snort Add-Ons

Output Viewer	URL	Description
		further at this point; we strongly recommend you use BASE instead.
BASE	http://sourceforge.net/projects/secureideas/	A later Web front end for Snort based off the ACID codebase, the Basic Analysis and Security Engine is our current favorite way to query and analyze Snort alerts.
Demarc	www.demarc.com	A commercial application that provides an interface similar to ACID's. It also requires Perl, and it is also strongly recommended that you encrypt the Demarc sessions as well.
Razorback	www.intersectalliance.com/projects/RazorBack/index.html	A GNOME/X11-based real-time log analysis program for Linux.
Incident.pl	www.cse.fau.edu/~valankar/incident	A Perl script used for creating incident reports from a Snort log file.
Loghog	http://sourceforge.net/projects/loghog	A proactive Snort log analyzer that takes the output and can e-mail alerts or block traffic by configuring IPTables rules.
Oinkmaster	www.algonet.se/~nitzer/oinkmaster	A tool used to keep your rules up-to-date.
SneakyMan	http://sneak.sourceforge.net	A GNOME-based Snort rules configurator.
SnortReport	www.circuitsmaximus.com/download.html	An add-on module that generates real-time intrusion detection reports.

Figure 4.5 Snort's Alerting Component.



Using Snort on Your Network

Your IDS can use just one Snort system, or more than one if you need redundancy or coverage of multiple network segments. For example, it is possible to divide the task of network monitoring across multiple hosts. The chief benefit of dividing tasks within a segment is redundancy—if one element of the system goes down, the network can still be monitored and protected. However, for monitoring extremely large and busy networks, we advise you to place at least one sensor in every distinct segment so that you can capture all the local traffic, not just the traffic that's sent to the segments where your main sensors are.

The previously outlined network structure can be used for *passive monitoring* or *active monitoring*. Passive monitoring is simply the ability to listen to network traffic and log it. Active monitoring involves the ability to either:

- Monitor traffic and then send alerts concerning the traffic that is discovered
- Actually intercept and block this traffic

Snort is primarily used for active monitoring and alerting, though it will generally not intercept and block unless you are using Snort inline and configure it accordingly.

Don't intrusion detection applications also do signature-based and anomaly-based detection? Signature-based detection means that you predefine what an attack looks like and then configure your network monitoring software to look for that signature. Anomaly-based detection requires the IDS to actually listen to the network and gather evidence about "normal" traffic. Then, if any traffic occurs that seems different, the IDS will respond by, for example, sending out an alert to the network administrator. Snort's rule-based matching is an example of signature detection, and some of the alerts generated by the preprocessors are examples of anomaly-based detection.

After dealing with a postmortem on a compromised system, you'll be amazed at how helpful a Snort NIDS can be. On the flip side, it's also frustrating when your Snort system does not log a possible attack. Let's take a possible attack: the IMAP login overflow attack. In this case, an attacker tries a buffer overflow to cause a remote root exploit.

Snort can let you know that someone is sending an IMAP packet that contains the signature of an IMAP login overflow. Depending on how you have Snort set up, you can either monitor the output or you can be notified by e-mail. Great, now you can yank the Ethernet cable from the wall and look at the corpse and find some tools used to break into the system and what they plan on doing on your machine.

The rule for detecting this attack is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 143 (msg:"IMAP login buffer \
      overflow attempt"; flow:established,to_server; content:"LOGIN";   \
      content:"{"; distance:0; nocase;                                    \
      byte_test:5,>,256,0,string,dec,relative; reference:bugtraq,6298;  \
      classtype:misc-attack; sid:1993; rev:1;)
```

This rule checks for any packet originating from the external network (defined by EXTERNAL_NET) to any system on the internal network (defined by HOME_NET) to port 143, which is the IMAP port. The *msg* variable defines what is sent to the Snort alert, and the rest of the information of the packet is content based. There are definitions on the type of attack (*misc-attack*), the SID number (1993), and the Bugtraq (www.securityfocus.com) reference on the attack *6298* (which you can find at www.securityfocus.com/bid/6298).

Then, there's the flip side: what happens when Snort does not detect an attack on your system? Take another UNIX system you have running. This one is running Apache with FrontPage extensions (gasp!). Someone finds a new overflow on FrontPage, writes a zero-day attack, and then he or she has your box. No IDS is perfect, and Snort will not catch attacks

if there's no preprocessor code or signature written to cover them yet. This is one of the primary reasons why it's important to keep your rules as up-to-date as possible—you stand a greater chance of detecting attacks if you have the most recent rules. Because rules actively developed as new attacks show up on the Internet, Snort's detection capabilities continually improve in response to the evolution of new attacks.

Snort's Uses

Snort has three major uses:

- A packet sniffer
- A packet logger
- An NIDS

All the uses relate to each other in a way that builds on each other. However, it's easiest to put the packet sniffer and the packet logger together in the same category—basically, it's the same functionality. The difference is that with the logging functionality, you can save the packets into a file. Conversely, you can read the packet logs with Snort as well.

Using Snort as a Packet Sniffer and Logger

In its simplest form, Snort is a packet sniffer. That said, it's the easiest way to start. The command-line interface for packet sniffing is very easy to remember:

```
# snort -d -e -v
```

Note that the *-v* option is required. If you run Snort on a command line without any options, it looks for the configuration file (.snortrc) in your home directory. Snort configuration files are discussed in Chapter 5.

Table 4.2 lists Snort options and their function.

Table 4.2 Basic Snort Options for Packet Sniffing and Logging

Option	What It Does
-v	Put Snort in packet-sniffing mode (TCP headers only)
-d	Include all network layer headers (TCP, UDP, and ICMP)
-e	Include the data link layer headers

You cannot use options *–d* and *–e* together without also using the *–v* option. If you do, you get the same output if you use *snort* without any options:

```
florida:/usr/share/doc/snort-doc# snort -de
Log directory = /var/log/snort
```

```
Initializing Network Interface eth0
using config file /root/.snortrc
Parsing Rules file /root/.snortrc

+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
ERROR: Unable to open rules file: /root/.snortrc or /root//root/.snortrc
Fatal Error, Quitting..
```

Now, if you run snort with the *–v* option, you get this:

```
whiplash:~ root# snort -v
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Verifying Preprocessor Configurations!
***
*** interface device lookup found: en0
***

Initializing Network Interface en0
OpenPcap() device en0 network lookup:
        en0: no IPv4 address assigned
Decoding Ethernet on interface en0

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.6.0 (Build 59)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/team.html
           (C) Copyright 1998-2006 Sourcefire Inc., et al.

01/22-20:27:44.272934 192.168.1.1:1901 -> 239.255.255.250:1900
UDP TTL:150 TOS:0x0 ID:0 IpLen:20 DgmLen:297
Len: 277
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/22-20:27:44.273807 192.168.1.1:1901 -> 239.255.255.250:1900
UDP TTL:150 TOS:0x0 ID:1 IpLen:20 DgmLen:353
```

```
Len: 333
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[]
```

After a while, the text scrolls off your screen. Once you press **Ctrl-C**, you get an output summary that summarizes the packets that Snort picked up, by network type (TCP, UDP, ICMP, IPX), data link information (including ARP), wireless packets, and any packet fragments.

```
Snort analyzed 56 out of 56 packets, dropping 0(0.000%) packets
Breakdown by protocol:      Action Stats:
    TCP: 0     (0.000%)        ALERTS: 0
    UDP: 44    (78.571%)       LOGGED: 0
   ICMP: 0     (0.000%)        PASSED: 0
    ARP: 1     (1.786%)
  EAPOL: 0     (0.000%)
   IPv6: 0     (0.000%)
    IPX: 0     (0.000%)
  OTHER: 11    (19.643%)
DISCARD: 0     (0.000%)
============================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0   (0.000%)
    Control Packets:    0   (0.000%)
    Data Packets:       0   (0.000%)
============================================================================
Fragmentation Stats:
Fragmented IP Packets: 0  (0.000%)
    Fragment Trackers: 0
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
  Frag2 memory faults: 0
============================================================================
TCP Stream Reassembly Stats:
     TCP Packets Used: 0     (0.000%)
     Stream Trackers:  0
     Stream flushes:   0
     Segments used:    0
```

```
     Stream4 Memory Faults: 0
===============================================================================
Snort received signal 2, exiting
```

Because this isn't very useful for checking the data of the packets, you'll run snort with the –*dev* option to give you the most information:

```
whiplash:~ root# snort -dev
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Verifying Preprocessor Configurations!
***
*** interface device lookup found: en0
***

Initializing Network Interface en0
OpenPcap() device en0 network lookup:
        en0: no IPv4 address assigned
Decoding Ethernet on interface en0

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.6.0 (Build 59)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/team.html
           (C) Copyright 1998-2006 Sourcefire Inc., et al.
01/22-20:28:16.732371 0:4:5A:F2:F7:84 -> 1:0:5E:7F:FF:FD type:0x800 len:0x5B
131.215.183.30:57535 -> 239.255.255.253:427 UDP TTL:254 TOS:0x0 ID:26121 IpLen:20
DgmLen:77
Len: 57
02 01 00 00 31 20 00 00 00 00 73 70 00 02 65 6E  ....1 ....sp..en
00 00 00 17 73 65 72 76 69 63 65 3A 64 69 72 65  ....service:dire
63 74 6F 72 79 2D 61 67 65 6E 74 00 00 00 00 00  ctory-agent.....
00                                               .

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/22-20:28:18.354830 0:4:5A:F2:F7:84 -> 1:0:5E:0:0:2 type:0x800 len:0x3E
131.215.184.253:1985 -> 224.0.0.2:1985 UDP TTL:2 TOS:0x0 ID:0 IpLen:20 DgmLen:48
Len: 28
```

```
00 00 10 03 0A 78 01 00 63 69 73 63 6F 00 00 00  .....x..cisco...
83 D7 B8 FE                                ....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

If you've used TCPDump before, you will see that Snort's output in this mode looks very similar. It looks very typical of a packet sniffer in general.

```
{date}-{time} {source-hw-address} -> {dest-hw-address} {type}
{length} {source-ip-address:port} -> {destination-ip-address:port} {protocol} {TTL}
{TOS} {ID} {IP-length} {datagram-length} {payload-length} {hex-dump} {ASCII-dump}
```

This is all great information that you're gathering, and Snort can collect it into a file as well as display it to standard output. Snort has built-in packet-logging mechanisms that you can use to collect the data as a file, sort it into directories, or store the data as a binary file.

To use the packet-logging features, the command format is simple:

```
# snort -dev -l {logging-directory} -h {home-subnet-slash-notation}
```

If you wanted to log the data into the directory /var/adm/snort/logs with the home subnet 10.1.0.0/24, you would use the following:

```
# snort -dev -l /var/adm/snort/logs -h 10.1.0.0/24
```

However, if you log the data in binary format, you don't need all the options. The binary format is also known as the TCPDump formatted data file. Several packet sniffers use the TCPDump data format, including Snort.

The binary format for Snort makes the packet collection much faster because Snort doesn't have to translate the data into a human-readable format immediately. You need only two options: the binary log file option *-L* and the binary option *-b*.

For binary packet logging, just run the following:

```
# snort -b -L {log-file}
```

For each log file, Snort appends a time stamp to the specified filename.

It's great that you're able to collect the data. Now, how do you read it? What you need to do is parse it back through Snort with filtering options. You also have the option to look at the data through TCPDump and Ethereal, as they use the same type of format for the data.

```
# snort [-d|e] -r {log-file} [tcp|udp|icmp]
```

The last item on the line is optional if you want to filter the packets based on packet type (for example, TCP). To take further advantage of Snort's packet-logging features, you can use Snort in conjunction with the Berkeley Packet Filter (BPF). The BPF allows packets to be filtered at the kernel level. This can optimize performance of network sniffers and log-

gers with marked improvements to performance. Because BPF filtering happens at a low level in the operating system, packets are eliminated from processing before they go through extensive processing at higher levels. To use Snort with a BPF filter, use the following syntax:

```
# snort –vd –r <file> <bpf_filter>
```

To help you find your feet, here are some examples of BPF filters. They are commonly used for ignoring packets and work with expressions (and, or, not).

If you want to ignore all traffic to one IP address:

```
# snort -vd -r <file> not host 10.1.1.254
```

If you want to ignore all traffic from the 10.1.1.0 network to destination port 80:

```
# snort -vd -r <file> src net 10.1.1 and dst port 80
```

If you want to ignore all traffic coming from host 10.1.1.20 on port 22:

```
# snort -vd -r <file> not host 10.1.1.20 and src port 22
```

For further information about BPF filters and their syntax, you can read the man page for tcpdump, which uses the same syntax (www.hmug.org/man/8/tcpdump.html).

Using Snort as an NIDS

Now that you understand the basic options of Snort, you can see where the IDS comes into play. To make Snort an IDS, just add one thing to the packet-logging function: the configuration file.

```
# snort -dev -l /var/adm/snort/logs -h 10.1.0.0/24 -c /root/mysnort.conf
```

Your rules are in the configuration file, and they are what trigger the alerts.

Snort and Your Network Architecture

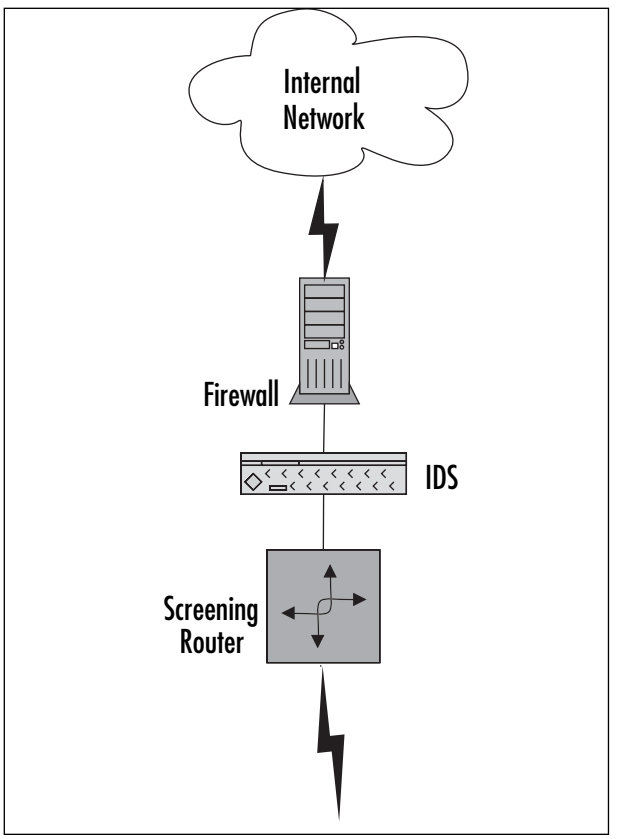
So how do you make Snort as useful as possible? You place your sensors as strategically as possible on your network, allowing them to see as much of the crucial network traffic as possible for your deployment. Where this is depends on several factors: how big your network is and how much money you can get your management to spend on Snort systems.

If you cannot get enough money to acquire enough Snort systems to achieve the optimal designs shown in Figure 4.6, you'll need to see what you can use from a practical sense. If you need to limit your spending, forego the system inside the router and just make sure you have the Snort systems inside the subnets you want to protect. In general, placing the sensors closer to your key assets will make it easier to see what those systems are sending and receiving. If you can't place sensors on all your subnets, choose wisely, and protect your most important machines with a sensor on their segments.

Many network administrators set up a screening router that acts as a poor-man's firewall and stops packets at the network level, usually by their well-known ports. The problem with this is that many packets can be rerouted through other ports.

However, if a packet does get past your screening router, it is useful to have an IDS sensor there to note the fact. The IDS sensor enables you to detect what you deem as attacks while enabling some filtering to hopefully catch some of the problems with the router. Figure 4.6 shows the IDS network architecture with a screening router.

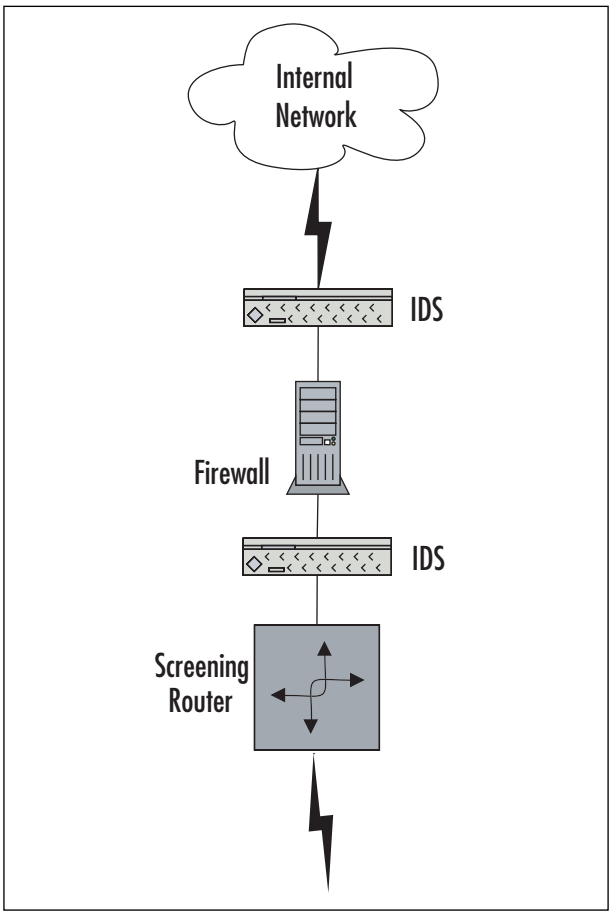
Figure 4.6 An IDS Network Architecture with a Screening Router



In this case, you would want to put an IDS system on the inside of your firewall and another in between your outside router and your firewall. Here, we're also assuming that your router is filtering some traffic through the access lists as well. You do not want your Snort system on the outside of your network because it will increase your false positive rate and leave your Snort system more vulnerable to attack (see Figure 4.7). Most important is the Snort system inside your firewall. This is the one you should monitor frequently for attacks. This system should trigger alerts only from potentially legitimate attacks and will produce many fewer false positives. However, the Snort system in between your router and

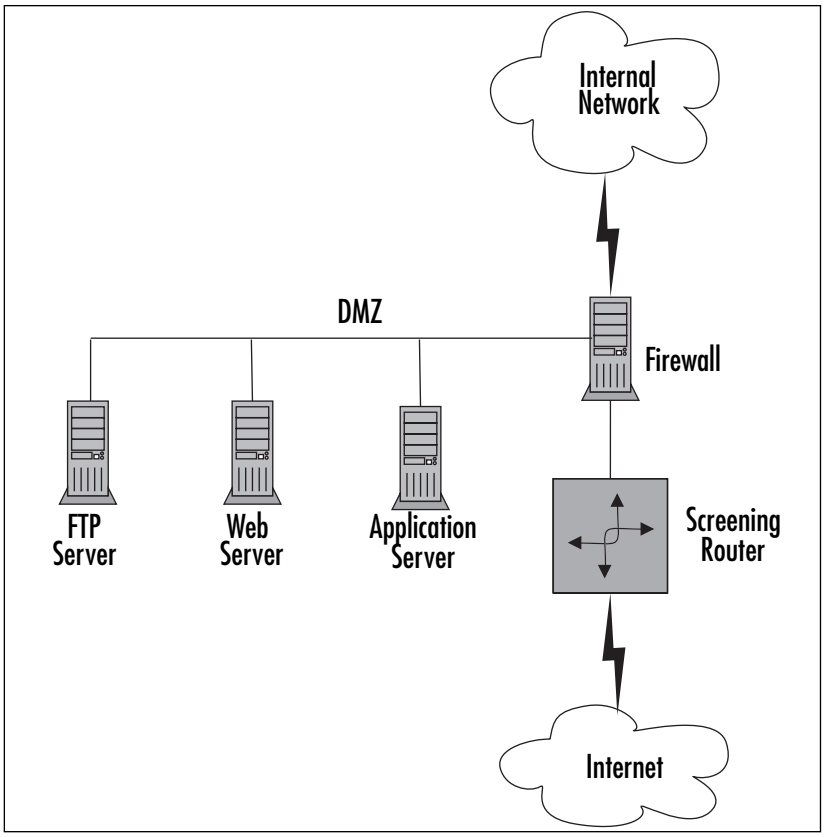
your firewall will also provide you with useful information, especially for a postmortem if one of your systems does get compromised.

Figure 4.7 A Firewalled Network with Snort Systems



Many network architectures have a demilitarized zone (DMZ) for providing public services such as Web servers, FTP servers, and application servers. DMZs can also be used for an extranet (which is a semitrusted connection to another organization), but we'll stick to the public server DMZ architecture in this example. This is illustrated in Figure 4.8.

Figure 4.8 A Firewalled Network with a DMZ

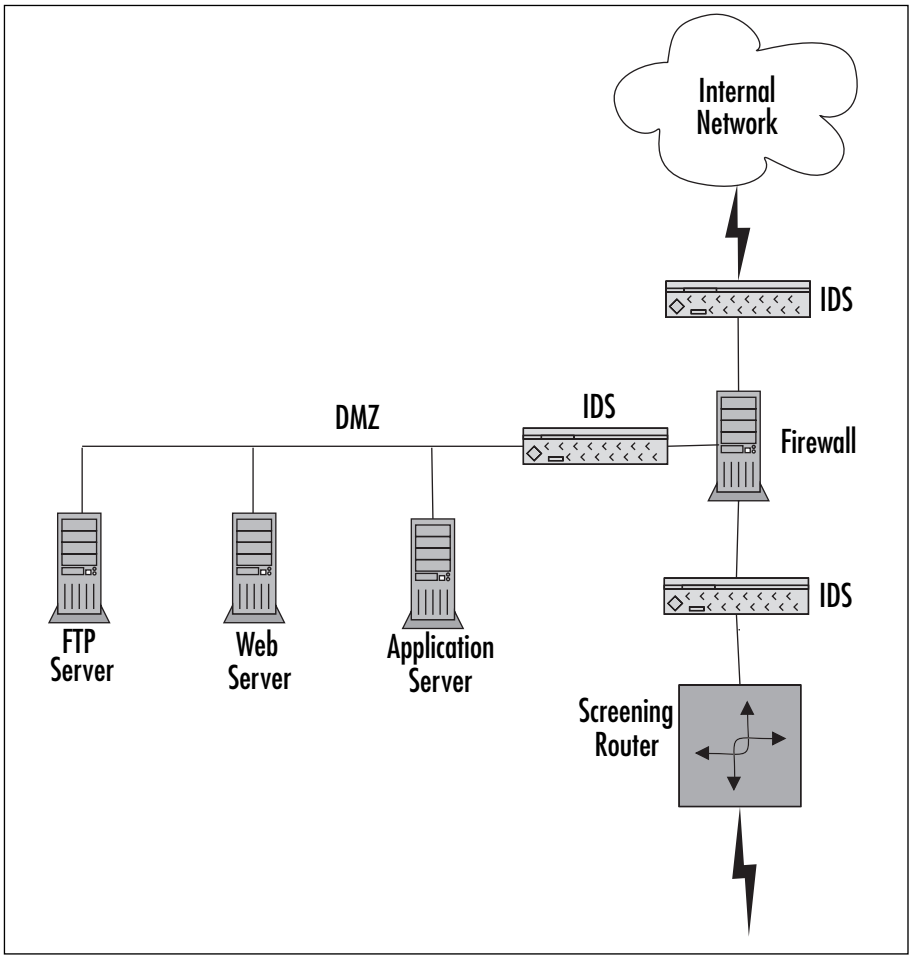


In this case, you would want three Snort systems: one inside the router, one inside the DMZ, and one inside the firewall. The reason for the additional IDS machine is because you have an additional subnet to defend. Therefore, a good rule of thumb for an optimal Snort deployment is:

- One inside the router
- One inside each subnet you want to protect

This is illustrated in Figure 4.9.

Figure 4.9 A Firewalled Network with a DMZ and Snort

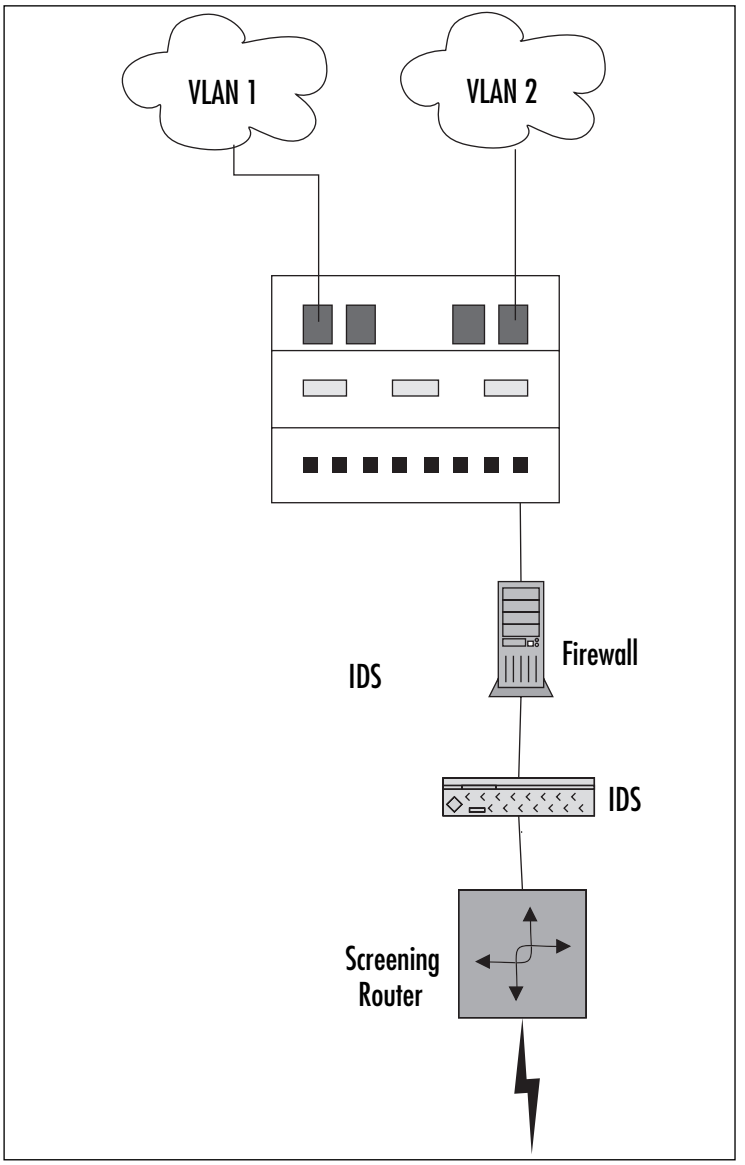


Snort and Switched Networks

Snort can be used on a switched network as well. Because switches are core infrastructure for most enterprises these days, monitoring them with Snort (or any other IDS) becomes more and more critical. Your switch can either be inside your router or inside your firewall.

A switch provides you with Layer 2 (Data Link layer on the OSI seven-layer model) configurability, including virtual LANs (VLANs), allowing you to subnet directly at the switch. Switches have also been used as overpriced routers. (You'll want to save your money if you're not using your switch's features.) In this case, you can connect the Snort system directly to the switch. The switch has a SPAN port (Switched Port Analyzer) port, which is where the Snort system will be connected. The Snort system then takes "copies" of all the packets to be analyzed, which are passed to it by the switch (see Figure 4.10).

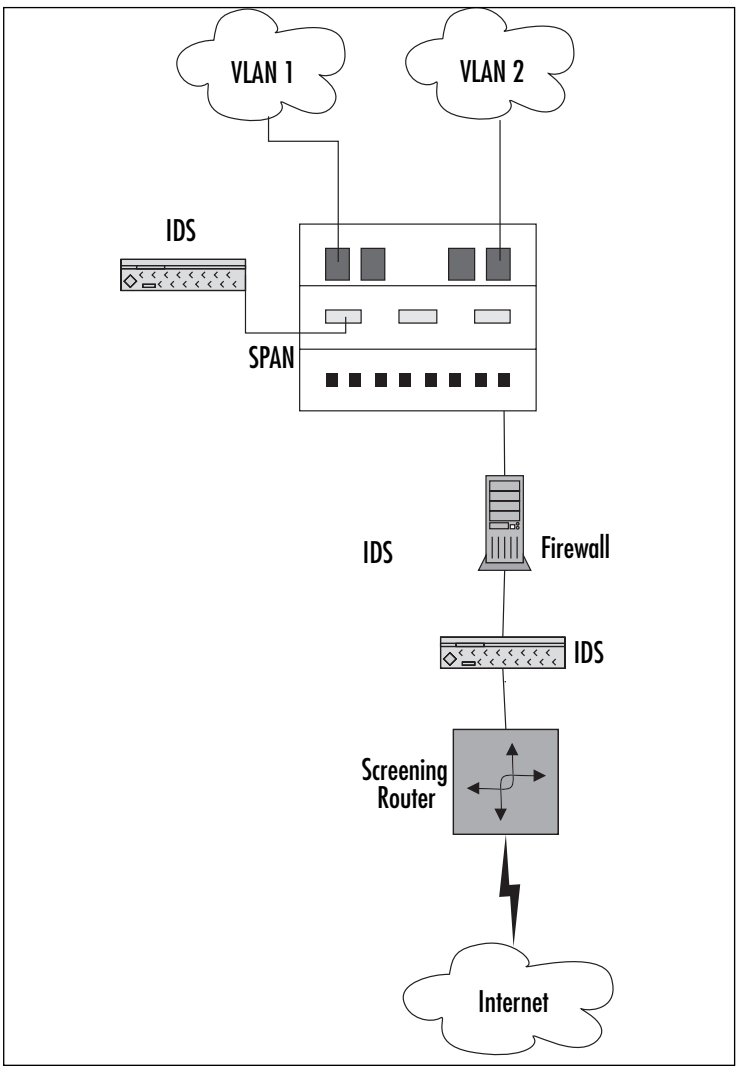
Figure 4.10 A Switched Network



In this case, you'll have to decide which other ports on your switch you want to monitor with the SPAN port. You can monitor just one port, or you can forward all traffic from a VLAN or even all traffic from the switch to the SPAN port. If you take that last option, it is important to keep an eye on traffic levels and make sure that the SPAN port is not overwhelmed; a flooded SPAN port drops packets and can spike its processors. If you're trying to shove 10 ports running at 100Mb each through one port running at 100Mb, it won't work,

and you might kill the performance of both your switch and your IDS (see Figure 4.11). We will discuss architecture and sensor placement in Chapter 6.

Figure 4.11 A Switched Network with Snort Systems



Pitfalls When Running Snort

Snort is a wonderful tool; however, like all tools, it has its pitfalls. Snort has three major pitfalls:

- Not picking up all the packets
- False positive alerts
- False negative alerts

Snort might not pick up all packets because of the speed of the network and the speed of the promiscuous interface. Snort's performance can also depend on the network stack implementation of the operating system. Ensure that your underlying infrastructure is as high end as possible to support your Snort deployment. In addition, to ensure optimal performance, it's a good idea to run some known attacks against the network segment that Snort is monitoring and ensure that it caught everything that it should have. Problems with dropped packets can lead to particular confusion with stream and flow reassembly, as well as missing critical network data.

False Alerts

False positives are when Snort gives you a warning when it shouldn't. Basically, a false positive is a false alarm. If you go with a default ruleset with Snort, then you will definitely get many false alarms. Why do IDSes behave this way? Well, it's better to get false alerts and whittle them down through tuning than it is to miss data that might have been a critical attack. So a new Snort installation can trigger a lot of alerts until you decide what is relevant to your network. The more open your network is, the more alarms you'll want to monitor.

On the opposite end, you can get false negatives. In other words, someone compromises a Snort- monitored system and your Snort system doesn't detect it. You might think that this doesn't happen, but when you get an e-mail from another system administrator describing a suspicious activity and your Snort system didn't pick it up, well, this is a very real scenario, and it usually happens with either out-of-date rulesets or brand-new attacks for which signatures have not yet been written. Make sure you keep your Snort rulesets up-to-date.

Upgrading Snort

Upgrading Snort can be quite painful for two reasons: the ruleset syntax may change, and the interface to the alert logs may change. We have found both to be obstacles when trying to upgrade Snort systems, and they can be quite a pain to deal with, particularly when you didn't want to have to do a forklift upgrade. If Snort changes its architecture to increase performance (as happened with the Snort 2.0 upgrade), you may experience a painful upgrade to any custom rulesets or alert interfaces in now-deprecated syntax and interfaces.

In addition, there are administrative foibles that may be encountered while creating rules, while reading logs, and while analyzing logs. When writing your own rules, make sure that they do what you think they're going to do, and test them to make sure that they alert you when they're supposed to. Rule syntax is tricky sometimes, and all it takes is one misplaced PCRE expression to cause either a whole lot of false positives or a whole lot of nothing. Having the rule in place won't help you much if the rule is incorrectly written. Similar attention should be paid when reading and analyzing logs—make sure that your security analysts understand the network and its context enough to be able to accurately identify when something is a false positive rather than a problem, and vice versa. We've seen

unfortunate deployments where clueless analysts marked every noisy rule as a false positive and tuned it out, rather than figuring out what was triggering the rule and writing a targeted pass rule for allowed traffic. That sort of approach doesn't help anyone, and may negate much of the benefit of having an IDS in the first place.

Security Considerations with Snort

Even though you are using Snort to improve your security, making sure that your Snort system is as secure as possible will make the data more trustworthy. If someone breaks into your Snort system, there is no reason to trust the alerts that it sends, thereby making the system completely useless until after you wipe the disks and reinstall everything.

Snort Is Susceptible to Attacks

With that said, a typical Snort installation is subject to attacks, both in Snort itself and in the underlying OS. Why? You'll want to get in remotely (SSH), and you'll probably want to store the alerts in a database (MySQL or Postgres). In addition, you'll probably want to view the alerts with a spiffy interface that might require a Web server (Apache or IIS). Any listening service is a possible surface for attacks, and some driver attacks can even target a listening interface that isn't advertising any services in particular at all. This makes your Snort system just like any other application, so stay on top of security vulnerability announcements and OS security announcements for whatever platform you've chosen, just as you would for any other crucial network appliance.

Now, based on this information, you may have several ports open on your Snort system: SSH (port 22), HTTP (port 80), HTTPS (port 443), and possibly MySQL (port 3306) or Postgres (port 5432). Anyone with access to the network can use NMAP and port scan your sniffer directly on its nonpromiscuous interface. This is one of the major reasons that we advocate having a separate interface for management than for sniffing and for locking down the management interface to restrict access and services as tightly as possible. Reducing the potential attack surface will help keep your IDS secure.

This is something that needs to be addressed because all of the preceding applications have had quite a few serious security issues, historically. In addition to making sure that your applications are up-to-date, you need to make sure that your kernel is configured properly and that it also is up-to-date. You didn't think that running Snort allows you to disregard basic system administration practices; did you?

Notes from the Underground….

Snort Security Vulnerabilities

All applications end up with some discovered vulnerabilities eventually. Snort is no exception. Although Snort itself has had relatively few flaws, some of the vulnerabilities in recent years have been notable. The RPC preprocessor flaw of 2003 (http://xforce.iss.net/xforce/alerts/id/advise141) allowed denial of service or potential host compromise. The flaw in the Back Orifice handling in 2005 (www.osvdb.org/displayvuln.php?osvdb_id=20034) could be triggered by a single UDP packet, and the frag3 Preprocessor Packet Reassembly Vulnerability earlier this year (2006) could potentially allow malicious traffic to pass undetected (www.osvdb.org/displayvuln.php?osvdb_id=23501). Because of issues like these, it is critically important to pay attention to vulnerability research and announcement lists and to patch your systems as new software becomes available.

Securing Your Snort System

Even though your Snort implementation is locked down, your system itself might not be. Make sure you do the basics. There are some things you need to do without exception:

- **Turn off services you don't need** Services like Telnet, the Berkeley R services, FTP, NFS, and NIS should not be running on your system. In addition, make sure you don't have any of the useless services running; for example, echo, discard, and chargen.
- **Maintain system integrity** Tripwire is a freeware application that checks for those backdoors and Trojans you don't suspect. There are plenty of other freeware applications like Tripwire—AIDE and Samhain are two worth mentioning.
- **Firewall or TCP Wrap the services you do use** Services like SSH and MySQL should be TCP wrapped or firewalled because they have their own security holes as well, and access should be restricted to the smallest possible set of necessary users. For services that you can't TCP Wrap such as Apache, make sure you have them configured as securely as possible. IPTables is the latest version of the Linux firewall, and there are plenty of references on how to implement it.
- **Encrypt and use public key authentication as much as you can** You should enable public key authentication only for OpenSSH. Another thing you might want to consider doing for Apache for using it to view logs is to use

Apache-SSL and use digital certificates for client-side authentication. This helps keep the obvious people out of your system through the usual compromisable channels.

- **Patch, patch, patch** We cannot stress this enough. Make sure you keep your patches and packages up-to-date as much as possible. Stay on top of applications you use and their security announcements—the same goes for any operating system you use. For FreeBSD/NetBSD/OpenBSD, make sure you keep your ports and packages up-to-date. For Red Hat Linux, make sure you stay on top of the updated RPMs. For those of you who are using Debian, you'll have the easiest time as long as you remember to run *apt-get update && apt-get upgrade* on a regular basis.

You can find more detail about securing your Snort system in Chapter 5.

Notes from the Underground….

Hardening Systems

You can perform all these actions on your own, or you can use something handy like Bastille Linux (www.bastille-linux.org/) to do the majority of the necessary hardening for you. (See Chapter 2 for more on Bastille.)

Summary

This chapter provided practical knowledge of the open-source IDS called Snort, and how it can help you with your security concerns. You learned about the history of Snort, how the Snort architecture works, and system requirements.

Additionally, you learned about Snort's different uses, including using Snort as a packet sniffer, a packet logger, and an IDS. You also learned about some pitfalls with Snort, including false positives.

Finally, this chapter also touched on some security issues that you should consider when running a Snort system. It's critical to keep the system as secure as possible, especially as an active packet logger or IDS.

Solutions Fast Track

What Is Snort?

- ☑ Snort is a packet sniffer, a packet logger, and a network IDS.
- ☑ We highly recommended having a large hard disk for data storage. Additionally, it is recommended to have two network interfaces on the system: one to run in promiscuous mode and the other for typical network connectivity (for example, SSH and HTTPS).

Exploring Snort's Features

- ☑ Snort's major components are the preprocessor, the detection engine, and the alert/logging components. All of Snort's components are implemented as plug-ins to increase flexibility.
- ☑ The preprocessor is used to take the packet data and process it before the data gets checked against the rules in the detection engine.
- ☑ The detection engine works by checking the data in each packet against a ruleset. Snort comes with a standard set of rules, but administrators can write their own as well.
- ☑ The alert/logging component takes the output of the data after it gets checked against the ruleset. The data can go straight into a log file in text or binary (TCPDump data) format. In addition, the data can be stored in SQL databases or be sent over the network through SNMP traps or WinPopup messages.

Using Snort on Your Network

- ☑ Snort can be used in various ways on your network. You can use it as a packet sniffer or as a packet logger in addition to for network intrusion detection.
- ☑ Snort can write packets in both text and binary mode. Binary mode is also known as TCPDump data format. This is not human readable, but it is a standard that Snort, TCPDump, and Ethereal all use to read and write network data. In addition to writing data, Snort can also filter the data to human-readable format from the binary format.
- ☑ Snort as an IDS needs to go on each of the private subnets you plan to monitor. It also helps to be able to place a Snort system behind the screening router as well.

Security Considerations with Snort

- ☑ Like any other application, Snort is subject to security vulnerabilities, including buffer overflows and DoS attacks.
- ☑ Snort should be upgraded on a regular basis to keep up-to-date with the latest signatures and the latest bug fixes with the application itself.
- ☑ In addition to securing the Snort application, you also need to secure the OS. This includes disabling unnecessary services, regularly applying patches, and proper configuration. It also includes encrypting sensitive traffic, such as login sessions with SSH and HTTP traffic with SSL.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Why log the Snort data in binary format? What can I gain from this?

A: Snort's binary format is also known as the TCPDump data format. Logging the packets to binary format makes packet collection faster. It also means that later you can look through the data and filter it after collection instead of during. Logging in binary format saves time because Snort does not have to translate the data from binary to human-readable format on the fly.

Q: How does Snort use plug-ins?

A: Snort uses plug-ins in various ways. The preprocessor can take plug-ins to translate data such as HTTP data into a more readable format, or it can take plug-ins that check for patterns such as checking for port scans. The detection engine can take rulesets of various types, but it can also take plug-ins. The alerting/logging component is the most obvious place you'll see plug-ins. The plug-ins for alerting/logging include functionality for SQL databases, SNMP traps, and WinPopup messages.

Q: How do I keep my Snort system secure?

A: Keeping your Snort system secure is just a matter of good system administration. This includes proper configuration, disabling unnecessary services, regular updates, and encrypting sensitive data.

Chapter 5

Installing and Configuring Snort and Add-Ons

Solutions in this chapter:

- **Placing your IDS**
- **Configuring Snort on a Windows System**
- **Configuring Snort on a Linux System**
- **Other Snort Add-Ons**
- **Demonstrating Effectiveness**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Placing Your NIDS

When it comes to implementing a network intrusion detection system (NIDS) like Snort, the single biggest factor in its effectiveness is its placement within the network. The value of the NIDS is in identifying malicious traffic and obviously it can't do that if it can't see the traffic. This means you want to place the NIDS in a location to maximize the data it will see. In smaller environments where there may be only one switch or hub, this is a pretty simple decision. Depending on your objectives, you may place it inline with the Internet connection only, so that you are inspecting traffic only to or from the Internet. In a larger installation, you will need to place multiple network cards in the NIDS so that it can inspect traffic from several chokepoints in your network.

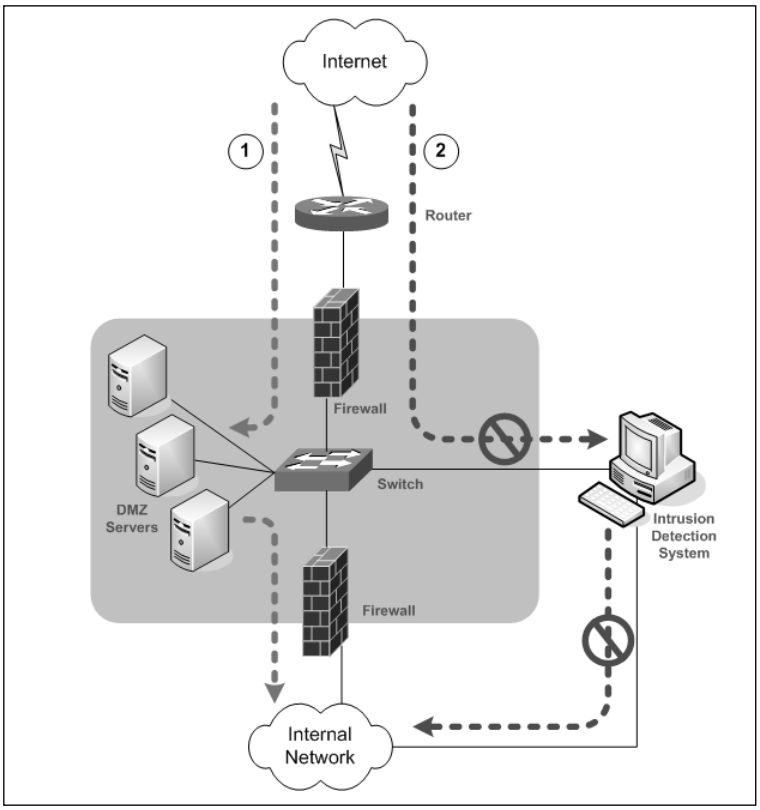
Notes from the Underground…

Further Considerations

Remember that an IDS is also a target for a hacker just like any other system, and often even more so. As such, the IDS host system should be hardened and locked down as much as possible. In addition to being a target because it can alert administrators to their activities, the hacker might target the IDS system itself because it often contains logs with valuable information in it on various systems. The IDS also has the capability of capturing packets that match its rulebase, and these packet dumps can contain valuable data as well. Don't neglect securing your IDS or you may be creating a security liability instead of the asset you intended.

Be cognizant of the fact that if you do choose to install multiple network cards to monitor multiple segments that you have the potential to create an alternate data path that enables traffic to bypass a firewall. As part of your hardening of the Snort host, you must ensure that routing is not enabled so that Snort cannot forward traffic from one segment that it is monitoring to another. There are multiple approaches to protect against this happening. The simplest is to use a network *tap* instead of just plugging in a normal network card. A tap is a specially designed piece of hardware that will only listen to traffic but will not transmit. Because it is hardware, there is no possibility of hacking the configuration or making a mistake in the configuration and accidentally allowing routing. Unfortunately, network taps are not free. Disabling routing, ensuring the host has no static routes, and disabling any routing protocols is the free way to ensure that you don't create a path around a firewall. Figure 5.1 illustrates bypassing the firewalls using your IDS.

Figure 5.1 Bypassing the Firewalls using the IDS



The first dotted line (data flow #1) represents the desired (secure) data flow. Traffic from outside can only terminate on a server in the DMZ, and traffic going into the internal network can only come from a server in the DMZ. With this configuration traffic from the Internet can never pass all the way through directly to a host on the internal network. The second data flow, #2, represents how an *incorrectly configured* IDS could be used to route traffic from the outside (untrusted) network, into the internal network.

When it comes to placement of your IDS, you need to be aware of the difference between a switch and a hub. A hub operates by sending any traffic it receives on any port to every other port. Therefore, when using a hub, the IDS will see all the traffic passing through that hub, which is usually what you want for your IDS. A switch is more advanced than a hub, and most new devices are switches. A switch listens and learns what machines are connected to which port. It then uses this information to construct a forwarding table. After it has learned which port a given host is on, it will then only send traffic destined for that host to that specific port. This means that without any additional configuration, when you plug your IDS into a switch port, it isn't going to be seeing much traffic.

Luckily, there are some options for getting around this feature. Most enterprise switches have a port mirroring option. The terms used to describe this functionality varies from one manufacturer to another, Cisco calls it *Switched Port Analyzer* (SPAN). This enables you to configure a specific port such that it will see traffic from other designated ports (or all other ports) even though the traffic is destined for a different port. Typically, one port is configured to mirror all other ports and the IDS is attached to this port. On a Cisco 3750 switch with 24 ports you could configure mirroring by entering the following commands:

```
switch(config)# monitor session 1 source interface gig1/0/1 - gig1/0/23
switch(config)# monitor session 1 destination interface gig1/0/24
switch(config)# end
```

This setup is pretty straightforward. Line one specifies which ports to forward traffic from, and line two specifies which port the traffic should be mirrored to. You will need to refer to the user guide for your specific switch hardware to see if port mirroring is supported, and if it is, how to configure it.

Configuring Snort on Linux

You will need to download the latest version of Snort that is appropriate for your system. If you are using Fedora Core 5, this is as simple as typing *yum install snort*, or you could download and install the .rpm from snort.org.

Configuring Snort Options

The next step is to configure the various options that determine how Snort will behave using the Snort configuration file. The configuration file is excellently documented and very easy to use. To get Snort working the way you want it to, follow these simple steps.

1. Start by opening the main Snort configuration file. By default it will be located at /etc/snort/snort.conf.
2. Configure the HOME_NET variable, if desired, by removing the # from the line you need. # is a commend indicator in the Snort configuration file. The HOME_NET variable defines which networks are the "trusted" internal networks. This is used with the signatures to determine when the internal network is being attacked. By default, HOME_NET is set to *any* network with the *var HOME_NET any* line in the snort.conf. Setting this to accurately reflect your internal address space will reduce the number of false positive alerts you receive. A common example would be *var HOME_NET 192.168.1.0/24* or perhaps *var HOME_NET [192.168.1.0/24,192.168.2.0/24]*.
3. Configure the EXTERNAL_NET variable if desired. This is the network you expect attacks to come from. The recommendation is to set this to everything

except your HOME_NET using the following: **var EXTERNAL_NET !$HOME_NET**. (Default: *var EXTERNAL_NET any*.)

4. Next, define what servers are running specific services. For example, by setting HTTP_SERVERS to only specific servers, Snort will only watch for HTTP attacks targeted at those servers. If you wish to see attacks targeting servers that are not running the affected services, leave the defaults, which are to watch for attacks directed towards *any* internal servers. (Default: *var DNS_SERVERS $HOME_NET*) If you had a Web server running on 192.168.1.11 and 192.168.1.12, you could tell Snort to only look for HTTP attacks targeting that server by setting the following variable: *var HTTP_SERVERS [192.168.1.11/32,192.168.1.12/32].*
5. If desired, configure the specific ports that services are available on. For example, the default for HTTP is defined on the following line: *var HTTP_PORTS 80.* Similar to defining the servers in the preceding section, this will tell Snort to only look for attacks targeting specific ports. With the default configuration, Snort would *ignore* an HTTP attack to port 8080. Again, this setting will help focus where Snort looks for different types of attacks to occur.
6. If you are interested in detecting the usage of AOL Instant Messenger (AIM), the various IP addresses of the AIM servers are defined in the snort.conf file. This is done because the IP addresses change frequently, and by using a variable, the rules don't have to be updated each time the IP address changes. If you don't wish to trigger based off AIM usage, don't worry about changing these IP addresses.
7. Download the Snort rules from http://snort.org/rules. Click **Download Rules** on the right-hand side of the page. On the **Download Rules** page, scroll down to the section labeled **Sourcefire VRT Certified Rules (unregistered user release)**. Download the latest ruleset.
8. Extract the rules (and /docs) to the location of your choice, typically /etc/snort/rules and /etc/snort/docs.
9. Configure the RULE_PATH variable, which tells Snort where to find the rules used for triggering events. You can use a relative path such as var RULE_PATH ../rules or an absolute path such as /etc/snort/rules.
10. The next section has some commented out lines to disable certain detections of some infrequently seen types of traffic. Unless you are having some issues with those alerts or your IDS is very low on resources, it's probably fine to just leave those at the default (enabled) configuration.
11. The next section enables you to configure the detection engine for systems with limited resources. Unless you are having issues, you can leave this option alone.

12. After that there are several sections of the configuration file to enable or disable specific functionality and detect particular types of attack, such as fragmentation attacks, stateful inspection, and stream reassembly options.
13. The section labeled Step #4 contains output options for Snort. Uncomment **output alert_syslog: LOG_AUTH LOG_ALERT** (the default). Despite what facility and severity you configure here, the snort alerts will be generated as auth.info. You also need to include the *–s* switch on the command line to enable syslog logging. We will discuss syslog in more detail in Chapter 8. If you don't have a syslog server to log to yet, just make note of the setting and come back to it when your syslog serer is set up.

- Using the preceding example of LOG_AUTH and LOG_ALERT, you would need the following in your syslog.conf file to log to a syslog server at 192.168.1.99:

```
auth. info          @managmentserverIP
```

- If you are using syslog-ng, you would need a logging destination defined, a filter that specifies what events to capture, and a log statement in the syslog-ng.conf file. An example of this configuration would be the following:

```
destination d_lab { udp ("192.168.1.99" port(514)); };
filter f_most { level(info..emerg); };
log { source(s_sys); filter(f_most); destination(d_lab); };
```

14. Edit the paths for the dynamically loaded libraries in section #2 to point to the proper path. Depending on your Linux distribution and installation method, these paths may not be the default. For example, on Fedora Core 5, using yum to install Snort, the settings would use the following paths: *dynamicpreprocessor directory /usr/lib/snort/dynamicpreprocessor* and *dynamicengine /usr/lib/snort/libsf_engine.so*. If you receive an error when you try to run Snort, along the lines of *Unknown rule type: dynamicpreprocessor directory* or *Unknown rule type: dynamicengine*, then your installation of Snort is not configured to use dynamically loaded processors. In this case, simply place a # in front of both of those lines to comment them out.
15. The last section (Step #6), contains various include statements that specify the rule-sets to be checked. Some rules are disabled by default, such as chat.rules, which is triggered by the use of various instant messaging clients. To enable or disable a given ruleset, simply add or remove a # at the beginning of the include line. This entry can be left as a relative path (for example, include $RULE_PATH/local.rules) because the RULE_PATH variable will be expanded to make it an absolute path.

16. If you need any custom rules that are not included with the standard Snort release, you can download rules provided by the Snort community from the Rules page on the Snort Web site. If you are looking for something unusual, you might find it there without having to create the rule yourself.

You are now ready to start up Snort and see what it looks like in action. When you start Snort you can specify the interface to listen on using the *–i* switch such as *–i eth0*. If you don't specify, it will use the first interface. Use the *–c* option to tell Snort which configuration file to use. It can be useful to have multiple configuration files configured so you can quickly switch configurations for special circumstances. You could prepare different configuration files to home in on certain issues, segments, or more in-depth logging. Another important option is *–A*, which tells Snort what type of alerts to generate. The options are fast, full, console, or none.

The following command example would start Snort listening on the first interface (no *–i* used), with alerts going to the console only, using the configuration file at /etc/snort/snort.conf. The *–l* switch tells Snort where the logging directory is located. The *–K* switch tells Snort what types of logs to generate. ASCII logs are easier for a human to read, but they take a little more time to log. If speed isn't a concern, the ASCII logs will probably be the easiest to read and analyze.

```
snort -A console -c /etc/snort/snort.conf -l /etc/snort/log -K ascii
```

You should see any triggered rules produce a message on the console and logged to your syslog server. If you add the –s switch to the end of the line, it will tell snort to log to the syslog server you have configured in the snort.conf file; however, it will not also display on the snort console. If you want to create a rule for testing purposes to see what the results look like, create a test rule file, such as TESTING.rules, and place it in the rules folder (/etc/snort/rules, in this example). In this file you could place the following line, which would trigger on any attempts to ping another system.

```
Alert icmp any any -> any any (msg:"TEST rule";)
```

Edit the snort.conf to read your new rule by inserting the following statement towards the end of the file: **include $RULE_PATH/TESTING.rules**. . As a last step, edit the snort\stc\sid-msg.map file. This file provides a mapping between snort alert messages and alert IDs or numbers. Custom alerts should use an ID number of more than one million. Add the following line at the end of the file:

```
1000001
```

Placing the ID number is the minimum requirement for Snort not to output an error. You can certainly fill in all the other fields, following the existing message maps as a guideline. When this is done, you will need to stop and restart Snort. Here is a partial display of the console output of a single ping and the reply.

```
10/12-21:29:35.911089  [**] [1:0:0] TEST rule [**] [Priority: 0] {ICMP}
192.168.1.99 -> 192.168.1.103
08/10-18:22:20.284438  [**] [1:0:0] TEST rule [**] [Priority: 0] {ICMP}
192.168.1.103 -> 192.168.1.99
```

You can also add your own custom rules to the local.rules file. When you open the file, you will find it is essentially empty, existing solely for you to place your custom rules in it. The local.rule is "included" in the snort.conf by default, so you will not need to add it there. You will, however, still need to edit the sid-msg.map file for any rules placed in local.rules.

The *–A* option will alter the display of the alerts on the console, while the *–K* option controls how the alerts are logged to the log directory. You should experiment with the different display formats to find the one that provides adequate information with the minimal strain on the Snort host. For day-to-day operations you would probably want to use fast alerts in your log files, which look like the ones that are sent to the console with the *console* option. Available alert modes and logging formats are outlined here for handy reference.

- **–A console** Logs to the console in the following format:

```
10/12-21:29:35.911089  [**] [1:0:0] TEST rule [**] [Priority: 0] {ICMP}
192.168.1.99 -> 192.168.1.103
```

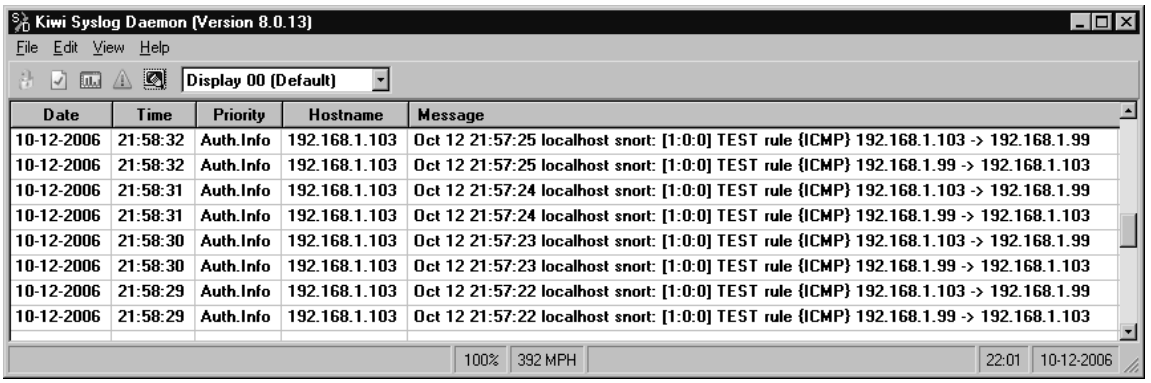
- **–A fast** Logs in the same *format* as console, but writes the alerts to a /snort/alert file with no output to the console.
- **–A full** Logs to the /snort/alert file in the following format:

```
[**] [1:0:0] TEST rule [**]
[Priority: 0]
10/12-21:38:53.741606 192.168.1.103 -> 192.168.1.99
ICMP TTL:64 TOS:0x0 ID:6350 IpLen:20 DgmLen:60
Type:0  Code:0  ID:512  Seq:7936  ECHO REPLY
```

- **–K pcap** This is the default mode if you don't specify an alternate format on the command line. This file will contain the alert packets in their entirety. You can open this file using a network sniffer such as Wireshark.
- **–K ascii** Will create a folder under /log for each IP address. Within that folder each rule will create a log file. The log entries will be the same format as the "full" alert format.
- **–K none** No log file will be created.

Congratulations! You now have a working IDS. Figure 5.2 shows the syslog alerts from the TESTING.rule in the Kiwi Syslog Daemon console.

Figure 5.2 Snort Alerts in Kiwi Syslog Daemon Console



Using a GUI Front-End for Snort

Like the Windows version of Snort, some have felt the administration of Snort could be improved upon by implementing a more robust GUI interface. There are several Snort GUIs to choose from aimed at both the configuration of Snort, as well as the interpretation of the Snort alerts. Some really only offer buttons to configure options on the Snort command line, and offer very little additional functionality, while others bring some very powerful additional features to the table. We will discuss the operation of some of the better offerings in the next section.

Basic Analysis and Security Engine

Basic Analysis and Security Engine (BASE) is available for download from http://base.secureideas.net/about.php. We'll get you up and running with BASE in this section, and then cover it in much more detail in Chapter 9. The purpose of BASE is to provide a Web-based front end for analyzing the alerts generated by Snort. Base was derived from the ACID project (Analysis Console for Intrusion Databases). Whereas ACID is more of a general-purpose front end for viewing and search events, BASE is a Snort-specific utility. The instructions to configure BASE assume you have already installed and configured Snort. Snort must be installed with the —*with-mysql* switch because Snort does not support MySQL output by default. The Snort Web site has RPM packages with MySQL support already included for some operating systems. This is the list of dependencies for running BASE: httpd, Snort (with MySQL support), MySQL, php-gd, pcre, php-mysql, php-pdo, php-pear-Image-GraphViz, graphviz, and php-adodb. Follow these steps to get BASE up and running.

1. Download and install MySQL and BASE
2. Edit the /snort/snort.conf file. Uncomment and edit the following line:

```
output database: log, mysql, user=snort password=snortpass dbname=snort
host=localhost
```

3. The next few steps are related to setting up the MySQL database and settings. After installing MySQL, enter the MySQL commands by typing **mysql** on the command line. This will place you in an interactive command mode. All commands must have a ; at the end of the line. By default, the MySQL installation will not have a password set at all. You should add a default password with the following commands.

```
mysql
mysql> SET PASSWORD FOR root@localhost=PASSWORD('somepassword');
```

After you have assigned a password to the root account, simply entering mysql will not enable you to access the interactive command mode. After a password has been assigned, use **mysql –u <username> –p**. You will then be prompted to enter the password for the user you specified (typically root).

4. The next step is to create the Snort database.

```
mysql> create database snort;
```

5. You now need to give the Snort user permissions to add the needed tables to the Snort database. Use these commands:

```
mysql> grant INSERT,SELECT on root.* to snort@localhost;
```

6. You should not set the password for the Snort user to the same password you used in the Snort configuration file.

```
mysql> SET PASSWORD FOR snort@localhost=PASSWORD('snortpass');
```

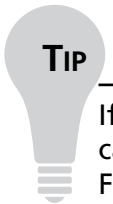
7. The next step is to add some additional permissions for the Snort database using the following commands:

```
mysql> grant ALL on snort.* to snort@localhost;
mysql> grant ALL to snort;
mysql> exit
```

8. Now that the database has been created, you need to populate it with the tables Snort uses. Use the following command to create the tables:

```
mysql –u root –p < /etc/snort/schemas/create_mysql snort
```

When the command completes, it will not give any indication of its success; therefore, it will be necessary to manually verify that the tables were created.



TIP

If the package you installed did not include the /snort/schemas/ directory, you can download the source package and extract the directory from there. With Fedora Core 5, for some reason installing the Snort with MySQL support did *not* include the schemas directory.

9. Verify the MySQL tables were created in the Snort database by entering the following commands. You should see output similar to that shown in the following example:

```
mysql –u root –p
show databases;
+----------+
| Database |
+----------+
| mysql    |
| snort    |
| test     |
+----------+
use snort;
show tables;
+------------------+
| Tables_in_snort  |
+------------------+
| data             |
| detail           |
| encoding         |
| event            |
| icmphdr          |
| iphdr            |
| opt              |
| reference        |
| reference_system |
| schema           |
| sensor           |
| sig_class        |
| sig_reference    |
| signature        |
| tcphdr           |
```

```
| udphdr           |
+------------------+
exit
```

The list of databases is not significant, as long as the Snort database exists, of course. The table listing must be accurate. If any are missing, Snort will generate an error when you run it.

10. Install **php-gd** which is used to generate the graphs in BASE. On Fedora Core 5 you can just type **yum install php-gd**.
11. Install ADODB, which is a database abstraction library for PHP. On Fedora you can simply enter **yum install php-adodb**.
12. It's now time to configure BASE itself. Edit the **/usr/share/base-php4/base_conf.php** file to ensure that the following lines are configured with paths and settings appropriate for your configuration.

```
$BASE_urlpath = '/base';
$DBlib_path = '/usr/share/ododb';
$DBtype         = 'mysql';
$alert_dbname   = 'snort';
$alert_host     = 'localhost';
$alert_port     = '';
$alert_user     = 'snort';
$alert_password = 'snortpass';
```

You should not be able to access the BASE Web page at the following URL: **http://localhost/base/**.

Tools & Traps…

Troubleshooting Tips

- You can enable debugging in BASE by editing the **/usr/share/base-php4/base-php4.conf** file.

```
$debug_mode = 2;
```

- Use chkconfig to make sure that MySQL, Snort, and httpd are running.

```
Chkconfig --list | grep snort
Snortd     0:off   1:off   2:on   3:on   4:on   5:on   6:on
```

Continued

If all entries say "off," then that service is configured not to start. Try **service snortd start**.

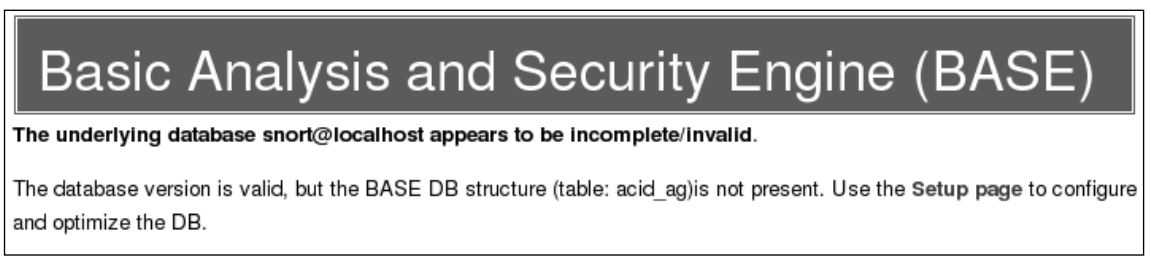
- Httpd may need to be restarted for some configuration changes to take effect; when in doubt, restart it just to be safe: **service httpd restart**.
- The httpd access log and error log can be found at /etc/httpd/logs.
- You can control the logging level of the httpd by editing /etc/httpd/conf/httpd.conf.

```
LogLevel debug
```

- If you are having issues with the URLs not being found, the /etc/httpd/conf.d/base-php4.conf file tells the Web server to alias /base/ with the directory /usr/share/base-php4/.

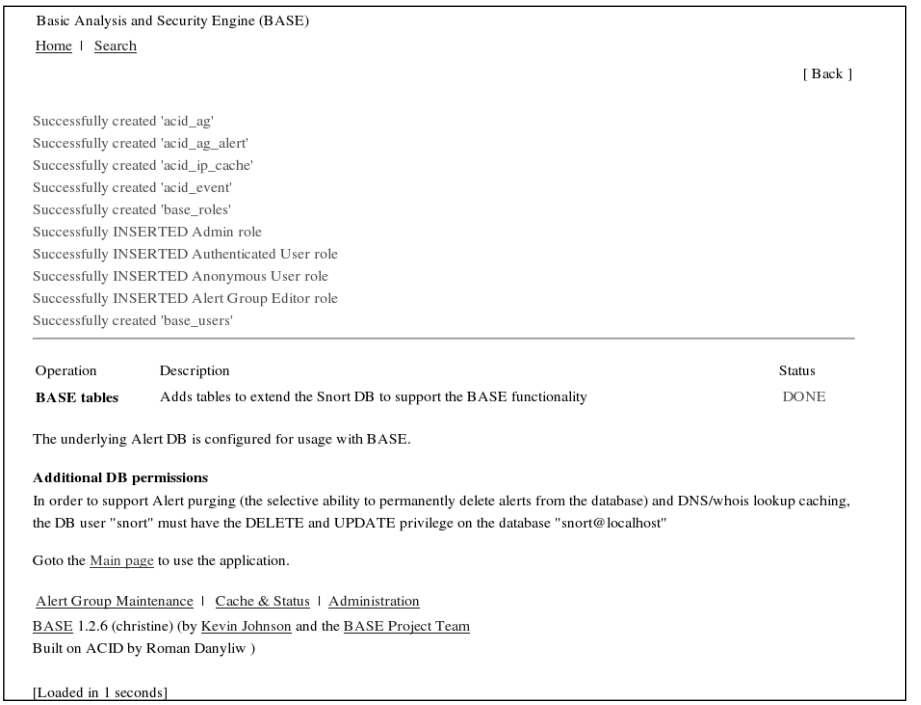
The very first time you start up BASE, none of the database tables have been created. You will see something like the page shown in Figure 5.3.

Figure 5.3 BASE Setup



13. Click on the **Setup page** link.
14. Click the **Create BASE AG** button on the right-hand side. You see several success messages as shown in Figure 5.4.

Figure 5.4 BASE Success



15. Click the **Main Page** link. This should take you to the primary BASE interface as shown in Figure 5.5.

Although this window may not be too flashy, there is a wealth of information you can discover. Most of the fields are actually links. By clicking to the right of **Today's alerts**, for example, you can get a sorted list of unique alerts, a listing of all alerts, or a list sorted by source IP address or destination IP address. The other headings along the left side offer similar functionality. Of particular note are the links for the **Most Frequent 15 addresses** by source address. This would enable you to quickly see which systems are *generating* the majority of your alerts. If you open that window (shown in Figure 5.6) there are several additional fields that are also hyperlinked.

Figure 5.5 BASE Main Page

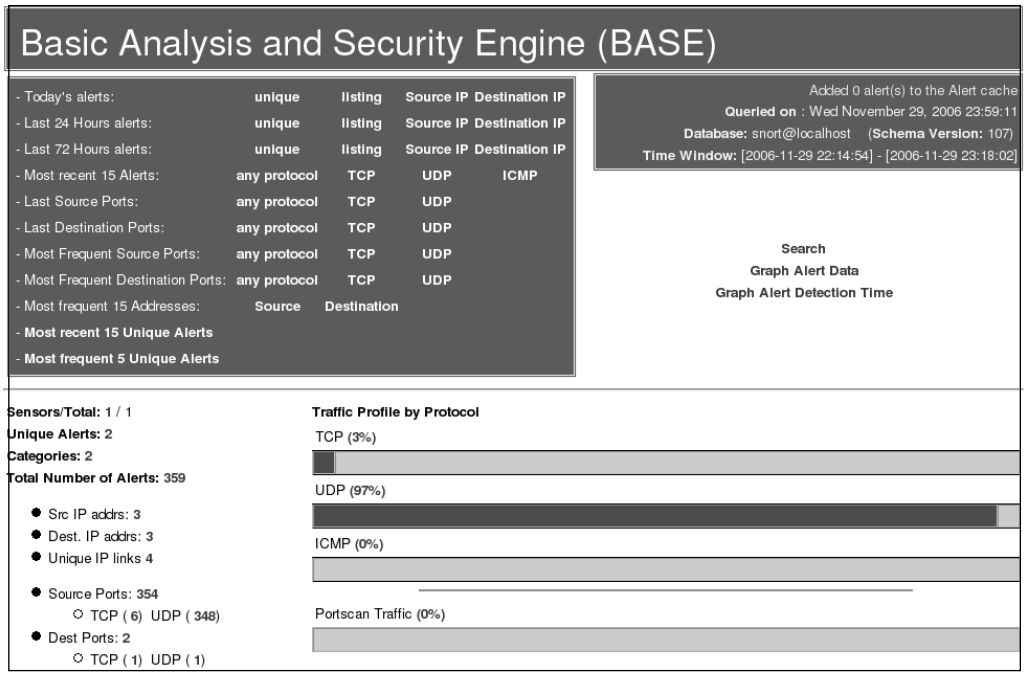
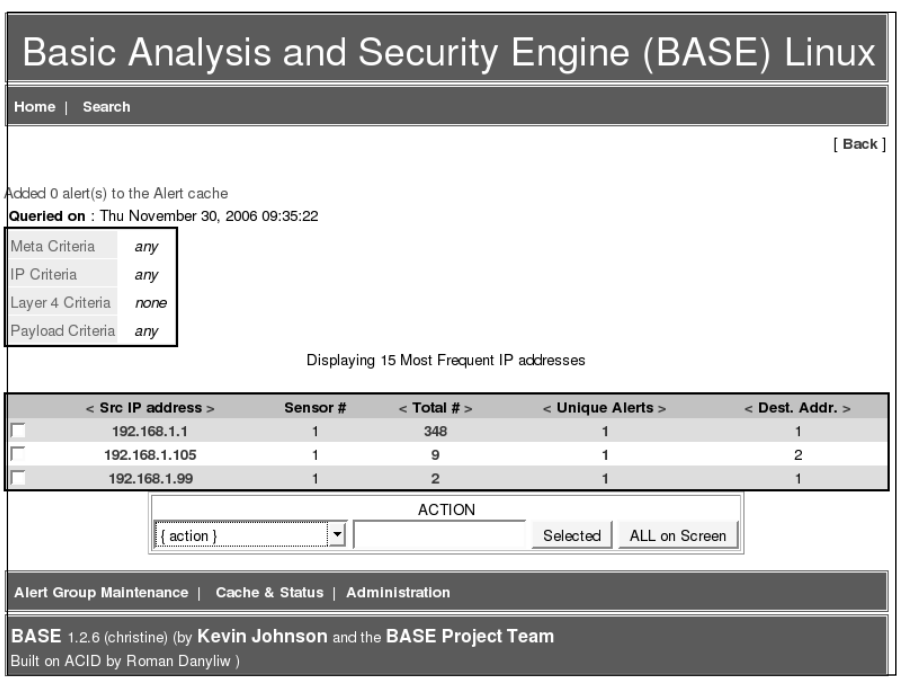


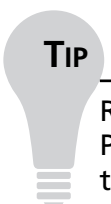
Figure 5.6 BASE Most Frequent by Source IP



Note the field at the bottom labeled **ACTION**. This enables you to configure the *alert groups*. Alert groups are basically shortcuts to enable you to view a subset of alerts quickly, without having to navigate through the various menus to get there. For example, suppose you want to know anytime that 192.168.1.1 generates an alert. You can check the check box to the left of 192.168.1.1, and then use the {action} drop-down box to select Create AG (by Name**)**. In the action column, enter .1_ALERTS to use as the alert group name. Finally, click Selected.

The next screen enables you to enter a description for the newly created alert group. Enter a meaningful text description for the group and click Save Changes. The next screen will be a listing of all alerts from 192.168.1.1. This screen *is* the alert group. In the future, if you want to quickly see this group of alerts, you can click Alert Group Maintenance at the bottom of each page, and then click the alert group you want to view. In this way, any subset of alerts is only two clicks away, sort of like a shortcut straight to a particular set of filtering criteria.

Another feature of note is the Administration link at the bottom of each page. This will take you to a screen where you can configure users for BASE. There are four options on the administration screen: list users, create a user, list roles, and create a role. These screens enable you to create users and assign them to various roles. If you click List Roles, you can see the four predefined roles. If you want to assign a user in the administrator role, simple click Create a user. Enter the login name, a full name or description, and a password. Use the drop-down box to select a role and then click Submit Query. None of the settings here will take effect until you edit the base_conf.php file and change the value of $Use_Auth_System = 1;. A value of 0 (the default) means the authentication is disabled and everyone has full access to BASE. Only the admin role has access to the administration screen.



TIP

Remember the different logging options for Snort on the command line. Previously we used *–A console*, which would log Snort events to the Snort terminal. If you are going to be using a different front end for viewing Snort alerts, there isn't much value in also logging to the console. You can use *–A none* when starting Snort, which will cause Snort not to log anything to the Snort terminal, resulting in improved performance.

Other Snort Add-Ons

The number of Snort utilities and add-ons is impressive. Some of these address such key issues as keeping your Snort rulebase up to date, while others provide additional perfor-

mance improvements such as faster logging. If you are looking for a particular feature or option, you should do some searching on the Internet, and you might find that the functionality you are looking for already exists. If you do find an add-in you are interested in using, remember to properly test it before deploying it in a production environment.

Using Oinkmaster

You may get tired of constantly having to update the Snort signature files. Because Snort is a signature-based IDS, having current signatures is vital. Without current signature files you could be unaware of intrusion attempts happening right in front of you. Although Snort itself does not include any means to automatically update the signature file, there is another utility that can help called Oinkmaster (http://oinkmaster.sourceforge.net/features.shtml). Oinkmaster is a Perl script that will update your Snort rules from the Snort Web site automatically. Because it uses Perl, Oinkmaster will run on a Linux or a Windows Snort host. The Oinkmaster Perl script can be scheduled to run and check for updates as often as you like. To get Snort rules downloads without having to wait until the next release of Snort, you have to register on the Snort Web site. You can register for free at https://snort.org/pub-bin/register.cgi. A password will be sent to the e-mail address you provide during registration. The configuration of Oinkmaster is outlined here.

1. After logging into the Snort Web site, click the link that says **User Preferences**.
2. At the bottom of the page is a section titled Oink Code; click the **Get Code** button.
3. **Copy** this code for use in the Oinkmaster configuration file.
4. Download the latest tar.gz from the Oinkmaster Web site.
5. Extract the folder in the archive to **/etc/oinkmaster**.
6. Edit the **oinkmaster.conf** file. Find the line that specified the URL for the current ruleset. (You can search for CURRENT.) Uncomment the line by removing the #, and then paste your oink code into the line in place of **<oinkcode>**.

```
url = http://www.snort.org/pub-bin/oinkmaster.cgi/<oinkcode>/snortrules-snapshot-
CURRENT.tar.gz
```

7. Start Oinkmaster with the following command:

```
oinkmaster.pl -C /etc/oinkmaster/oinkmaster.conf -o /etc/snort/rules
```

When it completes, Oinkmaster will tell you what rules were changed/updated. You can also specify the URL to retrieve the rules from the command line using the *–u <URL>* option. To configure the Oinkmaster script to run daily, use *crontab* with the following command:

```
crontab -u <user> -e
```

Enter the username you are running Oinkmaster as in place of <user>. This will open the crontab for that user. Adding the following line to the crontab will cause Oinkmaster to run each night at 2:00 A.M. If you prefer, there are also several GUI's available for configuring the cron daemon, such as gnome-schedule.

```
0 2 * * * oinkmaster.pl -C /etc/oinkmaster/oinkmaster.conf -o /etc/snort/rules
```

Now your Snort rules should stay up to date. Remember, if you change Snort versions, the URL to the appropriate rules may change, in which case you will need to update your oinkmaster.conf accordingly.



WARNING

Because the oinkmaster.conf file contains the path to update your Snort rules, if this file does not have secure access permissions on it, someone who could edit the file could render your IDS useless. With the ability to edit the configuration file, a malicious user could point the url to one of his choosing, with empty rule sets that will not trigger on anything, or even worse, rules that work perfectly except ignore the attacker's IP address. Make sure the oinkmaster.conf file is secured and only the account you are running Oinkmaster under has access to the file.

Additional Research

If the Snort utilities we have covered don't do everything you want them to do, there are other alternatives. Some of the utilities that are out there are more user friendly than others. Here are a few additional tools that are highly regarded and which may be helpful when running your Snort IDS. These include both Windows– and Linux-based tools.

- **ACID** ACID stands for Analysis Console for Intrusion Databases. You can download ACID from http://acidlab.sourceforge.net/. BASE was based off code from ACID, so the interfaces are strikingly similar. If you are only looking to use the Web front end for Snort logs, ACID probably doesn't buy you anything over BASE. If you plan to import data for additional non-Snort sources, however, ACID has the flexibility to do that.
- **Barnyard** Is available from http://sourceforge.net/project/showfiles.php?group_id=34732. It is basically a utility to offload the logging overhead from Snort. Using Barnyard, you configure

Snort to log binary data (which is the fastest way to Snort to log, but not very human-readable) and Barnyard will then take the binary logs and convert them to human-friendly ASCII or import them into a database. For a small environment with low-alert volumes on the IDS, Barnyard is probably not needed. Snort will support logging to a MySQL database natively without using Barnyard.

- **Sguil** Sguil (http://sguil.sourceforge.net/) is pronounced "sgweel" and stands for Snort GUI for Lamerz. It is also referred to as the Analysis Console for Network Security Monitoring. The objective of sguil is to provide more than just a console to view Snort alerts, but to also give the analyst the capability to delve deeper into an alert, all the way to the captured packet, to facilitate investigation. Basically, sguil integrates multiple security tools into one interface for easy access. The sguil developers provide a demo sensor that you can connect to from the Web to see sguil in action. To use it, simply download and install the sguil client, and then connect to the sensor demo.sguil.net on port 7734. When prompted, you can enter any user name and password, and then select the sensor names "reset" in the console. Sguil is a powerful tool for investigating Snort alerts, but the configuration and setup is not for the faint of heart.
- **Snortsnarf** This is a log analyzer targeted specifically at analyzing Snort logs. You can download it from www.snort.org/dl/contrib/data_analysis/snortsnarf/.

Demonstrating Effectiveness

One of the age-old debates when it comes to network data collection is placement of the sensors. This applies to both IDS sensors and reporting sensors such as PRTG Traffic Grapher. The most common difference of opinion is whether you should place the sensor outside your external firewall or inside it. This is relevant because the data you see will be drastically different between the two. With the sensor placed outside your perimeter firewall, you will see all traffic directed at you from the Internet, including all the traffic your firewall is blocking. If the sensor is placed inside the perimeter firewall, you will only see the traffic that has managed to pass through your firewall.

Undeniably, the traffic of the most security relevance is the traffic that has managed to traverse your firewall and get into your internal network. These are the potential attacks, probes, and whatnot that need to be inspected and monitored closely to make sure the network is not compromised. If everything is configured properly, an IDS inside the perimeter should really see very little traffic, except perhaps triggers related to IT policy, such as file sharing or instant messaging protocols. So if all the data a security officer would find "interesting" is on the inside, you might wonder what value a sensor on the outside would bring.

The best value for placing a sensor outside is really one of public relations. The unfortunate fact is that when it comes to network security, if everything is done properly, no one

ever sees much of anything. There are no flashing lights or alarms that say the network is functioning properly and securely. If you place an IDS on the outside of the perimeter, you can extract reports based on the traffic the IDS sees. These can be used to demonstrate to management in concrete terms what your security efforts are accomplishing. Saying "the network is running fine" is great, but probably doesn't have the impact that a one-page report with a pie chart would have. An executive summary of the attacks the sensor has seen could list some basic facts like "56,000 instances of code red worm were blocked, up 5% from last month," and so forth. With an old PC and a little up-front effort, these types of report would take very little effort to produce, but could reap huge rewards when it comes to public perception of network security.

When exposing *any* system to the Internet at large, remember it will be attacked. If your IDS is outside your perimeter firewall, there is nothing protecting the IDS except the IDS itself. This means the IDS will need to be hardened and secured as much as possible to ensure that it doesn't become a system for hackers to use. Under these circumstances, one of your best defenses would be for the IDS to use a network tap (not free) to ensure that it can only receive from the network and not transmit. There are various discussions on the Internet for making cables that can receive only. A little research will surely turn up some interesting designs to try. The success of these read-only cables will vary greatly depending on your system's network card and the switch or hub you are connected to. While this doesn't make the IDS sensor invulnerable to attacks or alleviate the need to harden it, this configuration will make it significantly harder to compromise.

Summary

Snort has the undisputed position as the lead open source IDS. As such, it enjoys several advantages. One advantage is the very large and diverse user base. This user base enables you to find a lot of help and information on the Internet for running, configuring, and customizing Snort. Although Snort may not enjoy the cohesive turnkey nature of a commercial package, you can assemble several utilities and tools to make Snort into an enterprise-class IDS. With no cost in software you can have an industry-standard IDS, with a large signature base and the ability to create your own custom signatures. You signatures can be automatically updated to keep them current, and you can use several GUI front ends to remotely configure and manage several Snort sensors at one central location. All this adds up to a lot of value and increased security, with no additional software cost.

Solutions Fast Track

Configuring an Intrusion Detection System

- ☑ Placement of the IDS will be key. If the IDS is not placed properly you will miss alerts and possibly think you are more secure than you really are.
- ☑ Your IDS is probably the security host that will need the most hardware resources of any discussed in this book (with the firewall being a close second), so plan accordingly when selecting the hardware to use for your IDS.
- ☑ Remember that even with the proper physical placement, you need to have a hub in order for the IDS to be able to see traffic destined for other devices, or enable port mirroring if you are using a switch instead of a hub.

Configuring Snort on a Linux System

- ☑ You may want to consider a Snort alert front end such as BASE for viewing alerts.
- ☑ If your environment is primarily Windows, this will enable you to access the alerts from the Windows systems without having to view the Snort console on the Linux IDS host.

Other Snort Add-Ons

- ☑ A fully functioning IDS will not be of much value if no one is taking notice of the alerts it generates. An easy-to-use alert console can add a lot of value to your IDS in that it may increase the attention the alerts receive.

- ☑ I recommend using Oinkmaster to automatically keep your Snort signature files current.

Demonstrating Effectiveness

- ☑ One of the age-old debates when it comes to network data collection is placement of the sensors.
- ☑ The most common difference of opinion is whether you should place the sensor outside your external firewall or inside it.
- ☑ Undeniably, the traffic of the most security relevance is the traffic that has managed to traverse your firewall and get into your internal network. These are the potential attacks, probes, and whatnot that need to be inspected and monitored closely to make sure the network is not compromised.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How do I configure Snort to send e-mail alerts?

A: You don't. Snort includes no native way to send e-mail alerts. This was an intentional decision because processing e-mail alerts would place an undue burden on the Snort process, possibly resulting in dropped packets and missed alerts. Instead, the simplest way to accomplish this is with a lag parsing tool, such as swatch.

Q: How do I turn Snort into an IPS instead of an IDS?

A: Snortsam (www.snortsam.net/) is designed to automatically adjust the rules on a firewall based on certain Snort alerts. It is a mature tool with relatively active development. Also check the user-contributed section of the Snort Web site for an assortment of utilities at www.snort.org/dl/contrib/patches/. Snort itself also has some limited capability to take actions, specifically when acting in "inline mode." Refer to the documentation at www.snort.org/docs/snort_htmanuals/htmanual_260/node7.html for more on Snort's native IPS support. See Chapter 11 for more on Snortsam.

FAQ

Q: How do I make a Snort rule to trigger for "X" application's traffic?

A: Start by searching online; you can usually find the rule already made for you. If not, the general procedure is to do a packet capture (with Wireshark, for example) and then review the packets. The tricky part is to identify something all the packets (or if not all, at least the initial packet) has in common. Some string that can uniquely identify that application's packet from any other's. Then you place this string in the rule using the payload option. See the online Snort manual for more information on rule option fields.

Q: How can I make my Snort sensor more secure?

A: There are many ways. First, configure a firewall on the sensor itself to protect itself. You would only filter traffic with a destination of the sensor, so that you don't accidentally filter the traffic you want to trigger alerts on. You can also have Snort listen on an interface without an IP address; this will make it a lot harder for an attacker to target the sensor. (See the main Snort FAQ for instructions on how to do this.)

Chapter 6

Advanced Snort Deployment

Solutions in this chapter:

- Introduction
- Monitoring the Network
- Configuring Channel Bonding for Linux
- Snort Rulesets
- Preprocessor Plug-Ins
- Detection Plug-Ins
- Output Plug-Ins
- Solving Specific Security Requirements

- ☑ Summary
- ☑ Solutions Fast Track
- ☑ Frequently Asked Questions

Introduction

You can make effective use of Snort by simply building and installing the stock source code and using the generic ruleset. However, if you are willing to write custom rules or write specialized plug-ins for augmenting preprocessing, detection, or postprocessing, then a new universe of possible uses presents itself. We will look at some of these possibilities by looking at various security requirements and how they might be implemented.

First let's look at just what these enhancing elements are and what kinds of things that they can do for us.

Monitoring the Network

There is always an issue on how to actually get monitored packets physically into Snort. With a network hub, it's just a matter of hooking it up to the Ethernet interface that you intend to use as Snort's monitor port. These days, hubs are found on only very small networks that do not have that much traffic on them. The real problem is what to do for a switched network. The question on how to monitor a switched network is one of the official Snort FAQs.

The answer depends on the type of switch you have, the capabilities of your Snort system, and how large your budget is. If you have a managed switch, you can usually put it into a mode where all traffic that crosses the switch can also be mirrored onto one of the switch ports (which is where you connect the Snort monitor interface). This configuration is called SPAN for Cisco switches; other manufacturers use terms such as port mirroring or monitoring for the same process. On some Cisco switches a similar mode called a VACL (VLAN Access Control List) can also be used. The use of this mode on a busy network can be problematic because it puts a heavy burden on the switch's CPU and can affect the performance of the switch. And, of course, if there is more than 1GB of total traffic crossing the VLAN (which is quite possible given multiple high-speed connections), then the span port won't be able to send all the traffic to Snort no matter what you do. If your network uses multiple VLANs, this can be a serious problem because you need to set up a separate SPAN port for each VLAN.

VLAN

Plugging a hub into the switch and plugging Snort into that is not a very good idea, except possibly for monitoring just the traffic that traverses a network (e.g., the traffic that moves into and out of a DMZ subnet, but not within the subnet). This is because the introduction of such a hub in an otherwise switched environment has a huge impact on the performance of the network.

Another alternative is to use a passive network tap. This hardware device aggregates the data on a switch and makes it available for monitoring. Passive network taps relieve the

switch of the burden of aggregating the data, and they offer users a good feature in that they are completely invisible to the monitored network. The problem with a network tap is that they require the Snort system to have two monitoring interfaces—one for each direction the data moves on the network. Now the two data streams need to be combined somehow to be able to get a complete picture of what is happening on the monitored network. Channel bonding (aka trunking), combining two network interfaces into a single logical one (this is more commonly done to provide improved network throughput) is one effective way to accomplish this combining of the data streams. Network taps are also a significant cost element in the implementation of an intrusion detection system (IDS).

Configuring Channel Bonding for Linux

Setting up channel bonding for Linux systems is straightforward. First, the kernel must be configured to support bonding. When you are configuring the kernel, the option for bonding driver support under the network device support menu should be set. If the bonding driver is set as a module (this is the easiest to manage), then there will be a module called bonding.o. Once the kernel supporting bonding is running or the bonding module is loaded, the interface can be started. For distributions that use configuration control like Red Hat, the following needs to be done to start the interface.

First, create a file ifcfg-bond0 in the directory /etc/sysconfig/network-scripts. This file should look something like:

```
DEVICE=bond0
BOOTPROTO=none
ONBOOT=yes
NETWORK=192.168.10.0
NETMASK=255.255.255.0
IPADDR=192.168.10.254
USERCTL=no
```

This sets up the virtual network interface, bond0 in this case. If the kernel is using bonding as a module, the modules.conf file should contain the entry alias bond0 bonding, which associates the module with this virtual interface. You still need to define which physical interfaces should be combined to implement the virtual interface. For eth0, the file ifcfg-eth0 should be edited to look like:

```
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
MASTER=bond0
SLAVE=yes
USERCTL=no
```

The configuration files for the other remaining interfaces (eth1, eth2, etc.) should be similarly edited. The configuration setup for other distributions is just as straightforward.

Snort Rulesets

Snort provides many rules as part of the stock installation. Even more rules and frequent updates are available by subscription at www.snort.org. Rather than covering just the basics, these rules cover many types of events, such as Web or SQL Server abuses, signatures that are indicative of viruses, the use of P2P protocols, and so on. When some new security threat emerges on the Internet, a new snort signature is frequently available within hours. In addition, writing your own rules is not that difficult.

You are probably already familiar with the basics of Snort rulesets; you set up a pattern that triggers an alert message if it is matched. For example,

```
alert icmp $EXTERNAL_NET any => $HOME_NET  any (msg:"ICMP JPEG data tunnel"; itype:
8; content:"JFIF"; classtype:string-detect; sid:1000000; rev:1;)
```

This rule will generate an alert when it detects an ICMP Ping packet that is carrying a JPEG image within it in an effort to transfer data without being detected (yes, you can do that, and yes it really happens).

The classic rules of this type can be set up to trigger an alert on a variety of conditions. This one uses the protocol and the packet content as a trigger. Other triggers include various TCP flags or IP options, the packet size, source, and destination. The really interesting rules have special functions that are invoked by using special keywords in the rule. For instance, the react trigger can be used to cause Snort to react to the packet in some manner, depending on what is desired. This trigger will attempt to close the connection and possibly send a visible warning to the originating system.



WARNING

If you use the react keyword, it should be the last one in the rule option list.

Other keywords in this class are resp, logto, session, and tag. The keyword resp is similar to react because it attempts to close the connection. However, resp can be configured to send a TCP reset to either the source or destination or ICMP unreachable messages to either end of the connection. Note that both resp and react attempt to close the connection. Depending on the network volume and the speed of your Snort system, Snort might not be able to react fast enough to succeed in shutting down that connection—especially if the connection is to a small Web page that is just a couple of packets long. Consequently, one should not rely on these keywords to assuredly close the connection.

The keyword logto enables you to send the packets that trigger the rule that contains this keyword to be logged to its own file. This keyword is ignored when Snort is in binary logging mode.

The keyword session, will follow a TCP session that is triggered by the rule and will allow logging of the entire connection. The keyword tag is similar to session, but once it is triggered, it follows all traffic from or to the source or destination, not just the current session, for a specified amount of time or number of packets. Both session and tag put a heavy burden on Snort and should be used for the post-processing analysis of Snort binary or tcpdump pcap files, not for routine real-time use. We will discuss using these keywords later in this chapter when we look at handling some specific security scenarios.

Starting with Version 2.6, you can also add/modify rules dynamically while Snort is running. These dynamic rules are written with a new C-like syntax. This new rule syntax is rather intimidating and is quite verbose when compared with the old syntax for simple rules. However, for complicated rules, the new syntax is easier to work with.

Here is an example that is shipped with the Snort 2.6.0 sources. This rule is for SID 637, and it shows the rule in the traditional syntax as a comment.

```
/*
 * sid637.c
 *
 * Copyright (C) 2006 Sourcefire,Inc
 * Steven A. Sturges <ssturges@sourcefire.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * Description:
 *
 * This file is part of an example of a dynamically loadable rules library.
 *
```

```
 * NOTES:
 *
 */

#include "sf_snort_plugin_api.h"
#include "sf_snort_packet.h"
#include "detection_lib_meta.h"

/*
 * C-language example for SID 637
 *
 * alert udp $EXTERNAL_NET any -> $HOME_NET any \
 * (msg:"SCAN Webtrends Scanner UDP Probe"; \
 * content:"|0A|help|0A|quite|0A|"; \
 * reference:arachnids,308; classtype:attempted-recon; \
 * sid:637; rev:3;)
 *
 */

/* content:"|0A|help|0A|quite|0A|";  */
static ContentInfo sid637content =
{
    "|0A|help|0A|quite|0A|",/* pattern to search for */
    0,                      /* depth */
    0,                      /* offset */
    CONTENT_BUF_NORMALIZED, /* flags */
    NULL,                   /* holder for boyer/moore info */
    NULL,                   /* holder for byte representation of "\nhelp\nquite\n"
*/
    0,                      /* holder for length of byte representation */
    0                       /* holder of increment length */
};

static RuleOption sid637option1 =
{
    OPTION_TYPE_CONTENT,
    {
        &sid637content
    }
};
```

```
/* references for sid 637 */
static RuleReference sid637ref_arachnids =
{
    "arachnids",    /* Type */
    "308"           /* value */
};

static RuleReference *sid637refs[] =
{
    &sid637ref_arachnids,
    NULL
};

RuleOption *sid637options[] =
{
    &sid637option1,
    NULL
};

Rule sid637 =
{
    /* protocol header, akin to => tcp any any -> any any */
    {
        IPPROTO_UDP,        /* proto */
        EXTERNAL_NET,       /* source IP */
        ANY_PORT,           /* source port(s) */
        1,                  /* direction, bi-directional */
        HOME_NET,           /* destination IP */
        ANY_PORT            /* destination port(s) */
    },
    /* metadata */
    {
        3,                  /* genid -- use 3 to distinguish a C rule */
        637,                /* sigid */
        3,                  /* revision */
        "attempted-recon",  /* classification */
        0,                  /* priority */
       "SCAN Webtrends Scanner UDP Probe",    /* message */
       sid637refs           /* ptr to references */
```

```
    },
    sid637options, /* ptr to rule options */
    NULL,                   /* Use internal eval func */
    0,                      /* Holder, not yet initialized, used internally */
    0,                      /* Holder, option count, used internally */
    0,                      /* Holder, no alert used internally for flowbits */
    NULL                    /* Holder, rule data, used internally */
};
```

Plug-Ins

Plug-ins are software modules that enable you to have custom functions that can run on the packets that are captured and decoded. To learn how to apply plug-ins, you first need to understand the overall data flow within Snort. When a packet is captured by Snort, it gets decoded and then passed on to higher modules. When Snort is run in Inline mode (for Linux), the capture engine is iptables instead.

The packets are next sent to the preprocessor module, the detection engine, and the output module. Each of these three modules can be supplemented with your own module plug-ins.

NOTE

Before Snort Version 2.6, you had to compile your plug-in directly into your Snort executable. With previous versions of Snort, in order to add a plug-in to your system, you had to rebuild the Snort binary, stop the old instance, install the new binary, and then start it. Beginning with Version 2.6, however, you can have *dynamic* plug-ins (if snort was compiled with the –enable-dynamic plug-in option set). With a dynamic plug-in, you build your plug-in, install it to the proper location on the file system, modify the configuration file (or the command line that you use), and then restart Snort. The API for the original static plug-ins and the dynamic plug-ins are slightly different. We focus here on the dynamic plug-ins.

Preprocessor Plug-Ins

Preprocessor plug-ins are analysis modules that run after the packet has been captured and decoded. These plug-ins are run on every packet that is received; the snort rules do not have any influence on them. These preprocessors can be used for various purposes, but they are

used typically in a context that is too complex to readily express in terms of a rule. Standard Snort includes several preprocessors, such as Frag3, Stream4, Flow, Portscan, and several for processing specific protocols (such as HTTP). Frag3 is for reconstructing fragmented packets before further analysis. Packets can become fragmented as a natural consequence of traveling across the network, but they can also be fragmented deliberately in an attempt to evade notice by simpler IDSes. Stream4 and Flow are for tracking a connection and following it through all the changes in state as it goes from open to established to closed.

Writing your own preprocessor plug-in is actually quite straightforward. For the dynamic plug-ins, there are three functions to write, and one of those is rather well fixed in its form. The first one registers the initialization function for the plug-in to Snort. Let's walk through the example from the Snort manual.

```
#define DYNAMIC_PREPROC_SETUP   ExampleSetup
extern void ExampleSetup();
extern DynamicPreprocessorData _dpd;

void ExampleInit(unsigned char *);
void ExampleProcess(void *, void *);

void ExampleSetup()
{
        /* register the init function with Snort */
        dpd.registerPreproc( "dynamic_example", ExampleInit );

    DEBUG_WRAP(_dpd.debugMsg(DEBUG_PLUGIN, "Preprocessor: Example is setup\n"););

}
```

Snort knows to call this particular function because the compiled module's symbol table equates DYNAMIC_PREPROC_SETUP to ExampleSetup. Snort will invoke ExampleSetup() and load these dynamic modules from the file system when it starts up. It is informed where to look by the `dynamicpreprocessor` keyword in the configuration file. For example

```
dynamicpreprocessor directory /usr/local/lib/snort_modules/
```

will cause all the modules in the directory **/usr/local/lib/snort_modules** to be loaded.

When Snort encounters a preprocessor setup line in the configuration file (e.g., preprocessor dynamic_example: port 123), it will run the function passed as the second argument of the registration function, in this case **ExampleInit()**. This function should parse any keywords that go with your preprocessor from the configuration file, and if it is satisfied with the configuration settings, it should also register the primary processing function.

```
u_int16_t portToCheck;

void ExampleInit(unsigned char *args)
{
    char *arg;
    char *argEnd;
    unsigned long port;

    _dpd.logMsg("Example dynamic preprocessor configuration\n");

    arg = strtok(args, " \t\n\r");

    if(!strcasecmp("port", arg))
    {
        arg = strtok(NULL, "\t\n\r");
        if (!arg)
        {
            _dpd.fatalMsg("ExamplePreproc: Missing port\n");
        }

        port = strtoul(arg, &argEnd, 10);
        if (port < 0 || port > 65535)
        {
            _dpd.fatalMsg("ExamplePreproc: Invalid port %d\n", port);
        }
        portToCheck = port;

        _dpd.logMsg("    Port: %d\n", portToCheck);
    }
    else
    {
        _dpd.fatalMsg("ExamplePreproc: Invalid option %s\n", arg);
    }

    /* Register the preprocessor function, Transport layer, ID 10000 */
    _dpd.addPreproc(ExampleProcess, PRIORITY_TRANSPORT, 10000);

    DEBUG_WRAP(_dpd.debugMsg(DEBUG_PLUGIN, "Preprocessor: Example is
initialized\n"););
}
```

Note that this function finally registered ExampleProcess() as the processing function. This function will get called for each packet that has been decoded. It is up to the plug-in to decide if the packet is to be processed or not. In this simple example, you process only TCP packets and will generate an alert whenever the port matches the one that was specified in the configuration file.

```
#define SRC_PORT_MATCH  1
#define SRC_PORT_MATCH_STR "example_preprocessor: src port match"
#define DST_PORT_MATCH  2
#define DST_PORT_MATCH_STR "example_preprocessor: dest port match"

void ExampleProcess(void *pkt, void *context)
{
    SFSnortPacket *p = (SFSnortPacket *)pkt;

    if (!p->ip4_header || p->ip4_header->proto != IPPROTO_TCP || !p->tcp_header)
    {
        /* Not for me, return */
        return;
    }

    if (p->src_port == portToCheck)
    {
        /* Source port matched, log alert */
        _dpd.alertAdd(GENERATOR_EXAMPLE, SRC_PORT_MATCH,
                      1, 0, 3, SRC_PORT_MATCH_STR, 0);
        return;
    }

    if (p->dst_port == portToCheck)
    {
        /* Destination port matched, log alert */
        _dpd.alertAdd(GENERATOR_EXAMPLE, DST_PORT_MATCH,
                      1, 0, 3, DST_PORT_MATCH_STR, 0);
        return;
    }
}
```

The static version of this same plug-in looks like the following,

```
extern void ExampleSetup();

void ExampleInit(unsigned char *);
void ExampleProcess(Packet *p);


void ExampleSetup()
{
        /* register the init function with Snort */
        RegisterPreprocessor( "static_example", ExampleInit );
}
```

This alert is directly analogous to the registration function in the dynamic version. The initialization function for the static version works the same way as well: it parses the arguments that may be passed and registers the main processing function. The static version of the initialization function also needs to register two additional functions—one that is invoked when a restart is initiated and the other that is invoked when Snort exits.

```
void ExampleCleanExit();
void ExampleRestart();
u_int16_t portToCheck;

void ExampleInit(unsigned char *args)
{
    char *arg;
    char *argEnd;
    unsigned long port;

    LogMessage("Example static preprocessor configuration\n");

    arg = strtok(args, " \t\n\r");

    if(!strcasecmp("port", arg))
    {
        arg = strtok(NULL, "\t\n\r");
        if (!arg)
        {
            FatalError("ExamplePreproc: Missing port\n");
```

```
        }

        port = strtoul(arg, &argEnd, 10);
        if (port < 0 || port > 65535)
        {
            FatalError("ExamplePreproc: Invalid port %d\n", port);
        }
        portToCheck = port;

        LogMessage("    Port: %d\n", portToCheck);
    }
    else
    {
        FatalError("ExamplePreproc: Invalid option %s\n", arg);
    }

    /* Register the preprocessor function */
    AddFuncToPreprocList(ExampleProcess);

    AddFuncToCleanExitList( ExampleCleanExit, NULL );

    AddFuncToRestartList( ExampeRestart, NULL );

}

void ExampleCleanExit()
{

}


void ExampleRestart()
{

}
```

The processing function itself is similar to the dynamic version as well,

```
void ExampleProcess(Packet *p)
```

```
{

    if (!p->ip4_header || p->ip4_header->proto != IPPROTO_TCP || !p->tcp_header)
    {
        /* Not for me, return */
        return;
    }

    if (p->src_port == portToCheck)
    {
        /* Source port matched, log alert */
        LogMessage("example_preprocessor source port (%d) match\n", p->src_port);
        return;
    }

    if (p->dst_port == portToCheck)
    {
        /* Destination port matched, log alert */
        LogMessage("example_preprocessor destination port (%d) match\n", p-
>dst_port );
        return;
    }
}
```

As we see here, the coding difference between the dynamic and static preprocessor plug-ins is minimal. The real difference is how they are "installed." As we saw earlier, to install the dynamic plug-in, you just point to where the compiled object file is located in the Snort configuration file and then restart Snort. For the static plug-in, you need to follow a much more invasive process. First, the primary functions must be declared in a header file (e.g., spp_example.h).

```
#ifdef SPP_EXAMPLE_H_
#define SPP_EXAMPLE_H_

void ExampleSetup();
void ExampleInit(char* args);

#endif
```

This file is then added to the list of includes in the file plugbase.c in the Snort source directory, src.

```
#include "preprocessors/spp_example.h"
```

Next, you need to add an invocation of the registration function, ExampleSetup(), to the list of similar function calls in the function InitPreprocessors().

Finally, you need to edit the Makefile in the preprocessors source file so that your new preprocessor source file is in the list of preprocessor files.

Then, of course, you need to recompile Snort and install the new binary. Clearly, this is a lot of work; hence, the motivation to develop a technique for loading plug-ins dynamically. Currently, the choice of approaches is available only for preprocessor plug-ins.

Detection Plug-Ins

Detection plug-ins are software modules that are run during the detection phase of processing the capture network packet. These rules *do* depend on the rulesets, or more accurately, they influence the rulesets. The standard installed detection plug-ins include ones for handling all the special rule keywords, such as react, and ones to handle checking various network packet header values, including the TTL, or comparing the stated data size with the actual data size. The Snort documentation and source code do not provide sample detection plug-ins, so to write one, you need to resort to reading the source code for the standard ones.

In this section we will show the essential elements of a detection plug-in, using sp_ttl_check as a model. The registration function looks as follows:

```
void TtlCheckInit(char* OptTreeNode *, int);


void SetupTtlCheck(void)
{
     RegisterPlugin( "ttl", TtlCheckInit);
}
```

This function is similar to what we have seen before, but the important difference is that it lets Snort know that there is a ruleset keyword named ttl. In addition, the function TtlCheckInit() will be invoked to parse the options for this keyword for each instance of the rules that use it. The TtlCheckInit() function will parse the options for the keyword and register what function to call for each option. In the actual implementation of the TTL check module, a lot of support code is not relevant to this discussion, but the important thing to remember is that the invocation of TtlCheckInit() ultimately leads to each valid option for the keyword registering itself to the list of options that Snort is to understand,

```
AddOptFuncToList( CheckTtlGT, on);
```

where the first parameter is the name of a function to call (in this case when the TTL keyword is passed a '>' parameter) and the second parameter is the OptTreeNode parameter that gets passed to TtlCheckInit().

Detection plug-ins are installed exactly the same way as statically built preprocessor plug-ins are.

Output Plug-Ins

Output plug-ins, also known as postprocessor plug-ins, run after the Snort detection engine. These plug-ins control where the result of the analysis will be sent. This can be a log file, a database, or a socket for communicating with another process. Currently, these plug-ins still have to be statically compiled into Snort when it is built; they cannot be dynamically loaded.

The database plug-ins are particularly useful for using Snort for special purposes. They allow most of the useful information about the network activity into an SQL database. The information can then be extracted and analyzed by other applications or for doing such things as a historical analysis of events on your network (e.g., when did we first start seeing port 0 probes?).

The API for output plug-ins is very similar to that of static preprocessor plug-ins. There is a function that registers the module and functions for the main processing, one for a clean exit, and one for a restart that must be registered. Documentation and examples for the output plug-ins do not explicitly exist; again, you need to look at the source code for the standard output plug-ins. It's really not as hard as it sounds; if you are comfortable with creating a static preprocessor plug-in, you will have no problems with writing an output plug-in if you really need to create a new one.

By appropriately combining custom rules, preprocessing plug-ins, detection plug-ins and postprocessing plug-ins, you can create a highly crafted Snort installation that can be utilized for many special purposes beyond the stock IDS function that it provides.

Snort Inline

Snort is traditionally used as an IDS, but it can also play an active role in your network security and be used as an intrusion *prevention* system. Snort must be specially compiled to use this mode, and you must be running on a Linux system that supports iptables. Once Snort inline is built and installed, you now have a system that can act as an integrated IDS and firewall system. When running in this mode, you have three additional rule types that you can use.

The first is drop, which will drop any packet that satisfies the rule. The second is reject, which will send a TCP reset or an ICMP unreachable message to the originator and drop the packet when the rule is triggered. Both of these forms will log the event in the Snort logs. A third rule type is sdrop, which will drop the packet without logging it.

The Snort inline mode is capable of replacing a packet in a limited way. The original packet and the new packet have to have the same length. Even so, this provides some inter-

esting possibilities for policy enforcement. For example, you could have a rule like the following example:

```
alert udp any any <> any 53 (msg:"udp replace"; content: "forbidden.com"; replace
"xxxxxxxxxxx"; )
```

This rule would prevent anybody from resolving the domain forbidden.com.

Solving Specific Security Requirements

Now that we have a familiarity with the different ways to enhance the utility of Snort, we will take a look at some specific security requirements and how we might use rules and/or plug-ins to address these issues.

Policy Enforcement

Security policy enforcement consists of two components: detection of violations and taking action when a violation occurs. Detecting violations is the traditional use of Snort. You can accomplish the detection through the use of an appropriate rule for the simpler cases or with a plug-in for more complicated policies. Taking action when a violation occurs depends on the local security policy and on how threatening the violation is. Action could be as simple as just logging the event, or it could involve an active response like those described earlier that could block the violating connection.

Catching Internal Policy Violators

Watching for internal systems that violate the local security policy is probably second only to watching inbound DMZ traffic in terms of usage for Snort. To use Snort for this type of monitoring, set up a series of rules that codify the local security rules, and then you are all set. For example

```
alert tcp $HOME_NET any <> $EXTERNAL_NET 1863 (msg:"CHAT MSN message";
flow:established; content:"MSG"; depth:4; content:"Content-Type|3A|"; nocase;
content:"text/plain"; distance:1 classtype:policy-violation; sid:540; rev:11;)
```

will trigger if an internal system connects to MSN messenger.

This type enforcement need is very common, so be sure to check to see if the ruleset that you are working already has what you need. Rules for watching chat protocols and P2P protocols tend to be rather complicated because many of these protocols can use multiple network ports or tunnel on ports used for other protocols (typically port 80, the HTTP port) so that they can easily get past firewalls.

Banned IP Address Watchlists

Probably the easiest way to watch for communication with banned IP addresses with Snort is by creating a set of rules for each address.

```
alert tcp $HOME_NET any <> $BANNED_NET any (msg:"TCP Traffic to banned network";
classtype:policy-violation; sid:1000001; rev:1;)
alert udp $HOME_NET any <> $BANNED_NET any (msg:"UDP Traffic to banned network";
classtype:policy-violation; sid:1000002; rev:1;)
```

The problem with rules like this is that if the list of banned networks (or IP addresses) is very large, it gets awkward maintaining a huge list of banned destinations. If you expect to be maintaining large lists of banned destinations, then a more elegant way of handling this is to store the banned list in a database and write a detection plug-in.

Network Operations Support

Using Snort for supporting network operations is primarily for making measurements of network utilization and performance.

Forensics and Incident Handling

Forensic issues present an interesting problem for the utilization of Snort. Generally, when we think about forensics we are dealing with analyzing what happened after the fact. In this case if you did not log the appropriate information or store it into a database, you are out of luck. However, if you are dealing with a situation that is ongoing, you can make good use of Snort's capabilities. Even with an ongoing incident, one cannot always know what to look for, so running Snort as an IDS with special rules or plug-ins is not the best way to do it. Instead, in this situation Snort should be run in a packet-sniffing mode, storing everything to a binary (pcap) file and *then* analyzing the captured data:

First,

```
snort  -deb –L logfile.pcap
```

then,

```
snort –r logfile.pcap –c special.conf
```

The configuration file special.conf will have the appropriate rules. In particular, this set of rules would involve the rule options: logto, session, and tag. So, for example, if you had reason to believe that something worthy of investigation was happening in a telnet session that involved the local system 192.168.100.78, then special.conf would contain a session rule like:

```
log tcp 192.168.100.78 any <> any 23 (session: printable; )
```

which will log all the printable characters in the telnet session from that suspect system.



WARNING

If you find yourself having to make this kind of packet sniffing and such deep packet analysis, make sure that *you* are not violating your organization's security and expectation of privacy policies by doing so. If you are an external security contractor doing this for a client, make sure that the client organization has given you explicit, legally binding permission, in writing, in order to conduct such an investigation. This "get out of jail free card" should include provisions that require the client to defend you in any lawsuit that arises as a result of your findings. You don't want any employee of the client who gets fired as a consequence of your investigation (we have seen this happen!) to respond by suing you! It's important to do this right. Hire a lawyer to help you and then make sure the client signs a permission form before you do any work.

Security incident handling is much like the forensics situation except that the goal is different. Instead of attempting to reconstruct the sequence of events and document everything that happened in the process, we are focused on finding out enough about the incident to mitigate the situation (although, it may turn out that you need to escalate the incident handling to a full-fledged forensic analysis).

Summary

The effectiveness and utility of Snort can be greatly expanded by combining the extra capabilities that arise by using custom rules and plug-ins. The process is not that hard, but the changes to the newest version of Snort provide you with a choice between the original method and the new way. The rulesets can now be written with two different syntaxes—the original traditional syntax and a new C-like syntax. The C-style syntax looks more verbose, but it has the advantage of being dynamically loadable. The richer syntax makes it easier to work with complicated rules. Preprocessing plug-ins are useful when packets are to be processed in a way that does not really fit well within rules, such as triggering an alert based on what has been seen across multiple packets. Preprocessing plug-ins can also be implemented in two different ways. The original method required statically compiling the plug-in directly into the Snort binary; the new method allows Snort to find and load the plug-in on the fly. The dynamic method is much easier to develop because it does not require rebuilding Snort to add the plug-in. The detection and output plug-ins can be developed only as statically compiled modules.

Solutions Fast Track

Monitoring the Network

- ☑ If you have a managed switch, you can usually put it into a mode where all traffic that crosses the switch can also be mirrored onto one of the switch ports (which is where you connect the Snort monitor interface).
- ☑ Plugging a hub into the switch and plugging Snort into that is not a very good idea, except possibly for monitoring just the traffic that traverses a network
- ☑ Passive network taps relieve the switch of the burden of aggregating the data, and they offer users a good feature in that they are completely invisible to the monitored network.

Configuring Channel Bonding for Linux

- ☑ When you are configuring the kernel, the option for bonding driver support under the network device support menu should be set.
- ☑ If the bonding driver is set as a module (this is the easiest to manage), then there will be a module called bonding.o

- ☑ Once the kernel supporting bonding is running or the bonding module is loaded, the interface can be started.

Snort Rulesets

- ☑ Snort rulesets are the basic method for customizing a Snort installation
- ☑ The new C-style rule syntax provides the ability to dynamically load rules and make it easier to work with complicated rules.

Preprocessor Plug-Ins

- ☑ Preprocessor plug-ins run after the decoder has run, they run on every packet
- ☑ Preprocessor plug-ins are useful for dealing with scenarios which are too complicated to handle with rulesets
- ☑ Preprocessor modules can be statically compiled into Snort or they can be dynamically loaded

Detection Plug-Ins

- ☑ Detection plug-ins enhance the rules by adding the functions to invoke in order to implement additional keywords for the rules
- ☑ Custom detection plug-ins must be statically compiled into the Snort binary

Output Plug-Ins

- ☑ Output plug-ins are for creating special output mechanisms for Snort, standard ones include writing to various databases.
- ☑ Custom output plug-ins must be statically compiled into the Snort binary

Solving Specific Security Requirements

- ☑ Security policy enforcement consists of two components: detection of violations and taking action when a violation occurs.
- ☑ Watching for internal systems that violate the local security policy is probably second only to watching inbound DMZ traffic in terms of usage for Snort.
- ☑ Probably the easiest way to watch for communication with banned IP addresses with Snort is by creating a set of rules for each address.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: When should a preprocessor plug-in be used?

A: Preprocessor plug-ins run on all decoded packets. They are typically used to handle the analysis of traffic that is too complex for Snort rulesets.

Q: When should a detection plug-in be used?

A: A detection plug-in can be used to add new keyword behaviors to Snort rulesets to enrich the standard ruleset syntax.

Q: When should an output plug-in be used?

A: An output plug-in should be used to augment the output options for Snort alerts; for example, to write to some exotic database system that Snort does not ordinarily support.

Q: Can Snort be used with a firewall to create an active intrusion prevention system (IPS)?

A: Yes, Snort can be used in inline mode with Linux iptables.

Chapter 7

Network Analysis, Troubleshooting, and Packet Sniffing

Solutions in this chapter:

- **What is Network Analysis and Sniffing?**
- **Who Uses Network Analysis?**
- **How Does it Work?**
- **Protecting Against Sniffers**
- **Network Analysis and Policy**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

"Why is the network slow?" "Why can't I access my e-mail?" "Why can't I get to the shared drive?" "Why is my computer acting strange?" If you are a systems administrator, network engineer, or security engineer you have heard these questions countless times. Thus begins the tedious and sometimes painful journey of troubleshooting. You start by trying to replicate the problem from your computer, but you can't connect to the local network or the Internet either. What should you do? Go to each of the servers and make sure they are up and functioning? Check that your router is functioning? Check each computer for a malfunctioning network card?

Now consider this scenario. You go to your main network switch or border router and configure one of the unused ports for port mirroring. You plug in your laptop, fire up your network analyzer, and see thousands of Transmission Control Protocol (TCP) packets (destined for port 25) with various Internet Protocol (IP) addresses. You investigate and learn that there is a virus on the network that spreads through e-mail, and immediately apply access filters to block these packets from entering or exiting your network. Thankfully, you were able to contain the problem relatively quickly because of your knowledge and use of your network analyzer.

What Is Network Analysis and Sniffing?

Network analysis (also known as traffic analysis, protocol analysis, sniffing, packet analysis, eavesdropping, and so on) is the process of capturing network traffic and inspecting it closely to determine what is happening on the network. A network analyzer decodes the data packets of common protocols and displays the network traffic in readable format. A *sniffer* is a program that monitors data traveling over a network. Unauthorized sniffers are dangerous to network security because they are difficult to detect and can be inserted almost anywhere, which makes them a favorite weapon of hackers.

A network analyzer can be a standalone hardware device with specialized software, or software that is installed on a desktop or laptop computer. The differences between network analyzers depend on features such as the number of supported protocols it can decode, the user interface, and its graphing and statistical capabilities. Other differences include inference capabilities (e.g., expert analysis features) and the quality of packet decodes. Although several network analyzers decode the same protocols, some will work better than others for your environment.

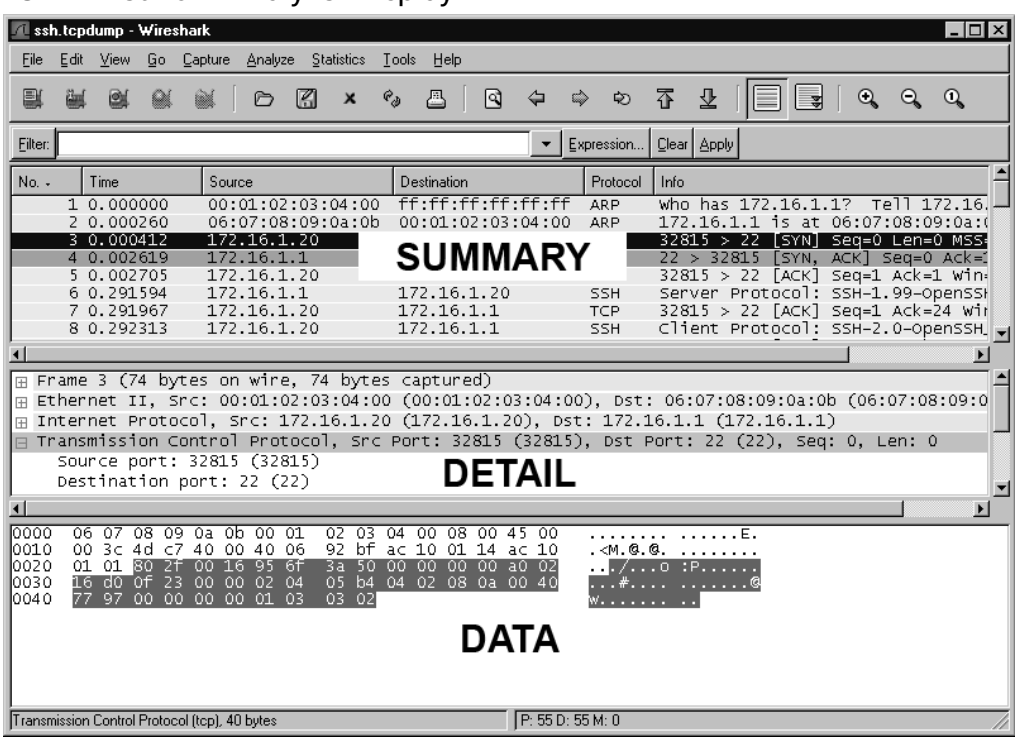
NOTE

The "Sniffer™" trademark, (owned by Network General) refers to the Sniffer product line. In the computer industry, "sniffer" refers to a program that captures and analyzes network traffic.

Figure 7.1 shows the Wireshark Network Analyzer display windows. A typical network analyzer displays captured traffic in three panes:

- **Summary** This pane displays a one-line summary of the capture. Fields include the date, time, source address, destination address, and the name and information about the highest-layer protocol.
- **Detail** This pane provides all of the details (in a tree-like structure) for each of the layers contained inside the captured packet.
- **Data** This pane displays the raw captured data in both hexadecimal and text format.

Figure 7.1 Network Analyzer Display



A network analyzer is a combination of hardware and software. Although there are differences in each product, a network analyzer is composed of five basic parts:

- **Hardware** Most network analyzers are software-based and work with standard operating systems (OSes) and network interface cards (NICs). However, some hardware network analyzers offer additional benefits such as analyzing hardware faults (e.g., cyclic redundancy check (CRC) errors, voltage problems, cable problems, jitter, jabber, negotiation errors, and so on). Some network analyzers only support Ethernet or wireless adapters, while others support multiple adapters and allow users to customize their configurations. Depending on the situation, you may also need a hub or a cable tap to connect to the existing cable.
- **Capture Driver** This is the part of the network analyzer that is responsible for capturing raw network traffic from the cable. It filters out the traffic that you want to keep and stores the captured data in a buffer. This is the core of a network analyzer—you cannot capture data without it.
- **Buffer** This component stores the captured data. Data can be stored in a buffer until it is full, or in a rotation method (e.g., a "round robin") where the newest data replaces the oldest data. Buffers can be disk-based or memory-based.
- **Real-time Analysis** This feature analyzes the data as it comes off the cable. Some network analyzers use it to find network performance issues, and network intrusion detection systems (IDSes) use it to look for signs of intruder activity.
- **Decode** This component displays the contents (with descriptions) of the network traffic so that it is readable. Decodes are specific to each protocol, thus network analyzers vary in the number of decodes they currently support. However, new decodes are constantly being added to network analyzers.

NOTE

Jitter is the term that is used to describe the random variation of signal timing (e.g., electromagnetic interference and crosstalk with other signals can cause jitter). *Jabber* is the term that is used to describe when a device is improperly handling electrical signals, thus affecting the rest of the network (e.g., faulty NICs can cause jabber).

Who Uses Network Analysis?

System administrators, network engineers, security engineers, system operators, and programmers all use network analyzers, which are invaluable tools for diagnosing and troubleshooting network problems, system configuration issues, and application difficulties. Historically, network analyzers were dedicated hardware devices that were expensive and difficult to use. However, new advances in technology have allowed for the development of software-based network analyzers, which make it more convenient and affordable for administrators to effectively troubleshoot a network. It also brings the capability of network analysis.

The art of network analysis is a double-edged sword. While network, system, and security professionals use it for troubleshooting and monitoring the network, intruders use network analysis for harmful purposes. A network analyzer is a tool, and like all tools, it can be used for both good and bad purposes.

A network analyzer is used for:

- Converting the binary data in packets to readable format
- Troubleshooting problems on the network
- Analyzing the performance of a network to discover bottlenecks
- Network intrusion detection
- Logging network traffic for forensics and evidence
- Analyzing the operations of applications
- Discovering faulty network cards
- Discovering the origin of virus outbreaks or Denial of Service (DoS) attacks
- Detecting spyware
- Network programming to debug in the development stage
- Detecting a compromised computer
- Validating compliance with company policy
- As an educational resource when learning about protocols
- Reverse-engineering protocols to write clients and supporting programs

How Are Intruders Using Sniffers?

When used by malicious individuals, sniffers can represent a significant threat to the security of a network. Network intruders use sniffing to capture confidential information, and the terms *sniffing* and *eavesdropping* are often associated with this practice. However, sniffing is

becoming a non-negative term; most people use the terms sniffing and network analysis interchangeably.

Using a sniffer in an illegitimate way is considered a *passive attack*, because it does not directly interface or connect to any other systems on the network. A sniffer can also be installed as part of the compromise of a computer on a network using an *active attack*. The passive nature of sniffers is what makes detecting them difficult. (The methods used to detect sniffers are detailed later in this chapter.)

Intruders use sniffers on networks for:

- Capturing cleartext usernames and passwords
- Discovering the usage patterns of the users on a network
- Compromising proprietary information
- Capturing and replaying Voice over IP (VoIP) telephone conversations
- Mapping the layout of a network
- Passive OS fingerprinting

The above are all illegal uses of a sniffer unless you are a penetration tester whose job is to find and report these types of weaknesses.

For sniffing to occur, an intruder must first gain access to the communication cable of the systems of interest, which means being on the same shared network segment or tapping into the cable somewhere between the communication path. If the intruder is not physically present at the target system or communications access point (AP), there are still ways to sniff network traffic, including:

- Breaking into a target computer and installing remotely controlled sniffing software.
- Breaking into a communications AP (e.g., an Internet Service Provider [ISP]) and installing sniffing software.
- Locating a system at the ISP that already has sniffing software installed.
- Using social engineering to gain physical access to an ISP in order to install a packet sniffer.
- Having an inside accomplice at the target computer organization or the ISP install the sniffer.
- Redirecting or copying communications to take a path that includes the intruder's computer.

Sniffing programs are included with most *rootkits* that are typically installed on compromised systems. Rootkits are used to cover the tracks of an intruder by replacing commands

and utilities and clearing log entries. Intruders also install other programs such as sniffers, key loggers, and backdoor access software. Windows sniffing can be accomplished as part of a Remote Admin Trojan (RAT) such as SubSeven or Back Orifice. Intruders often use sniffing programs that are configured to detect specific things (e.g., passwords), and then electronically send them to the intruder (or store them for later retrieval by the intruder). Vulnerable protocols for this type of activity include Telnet, File Transfer Protocol (FTP), Post Office Protocol version 3 (POP3), Internet Message Access Protocol (IMAP), Simple Mail Transfer Program (SMTP), Hypertext Transfer Protocol (HTTP), Remote Login (rlogin), and Simple Network Management Protocol (SNMP).

One example of a rootkit is "T0rnKit," which works on Solaris and Linux. The sniffer that is included with this rootkit is called "t0rns" and is installed in the hidden directory */usr/srec/.puta*. Another example of a rootkit is Linux Rootkit 5 (Lrk5), which installs with the linsniff sniffer.

Intruders commonly use sniffer programs to control back doors. One method is to install a sniffer on a target system that listens for specific information and then sends the backdoor control information to a neighboring system. This type of backdoor control is hard to detect, because of the passive nature of sniffers.

cd00r is an example of a backdoor sniffer that operates in non-promiscuous mode, making it even harder to detect. Using a product like Networked Messaging Application Protocol (Nmap) to send a series of TCP synchronous (SYN) packets to several predefined ports will trigger the backdoor to open up on a pre-configured port. More information about *cdoor* can be found at *www.phenoelit.de/stuff/cd00r.c*.

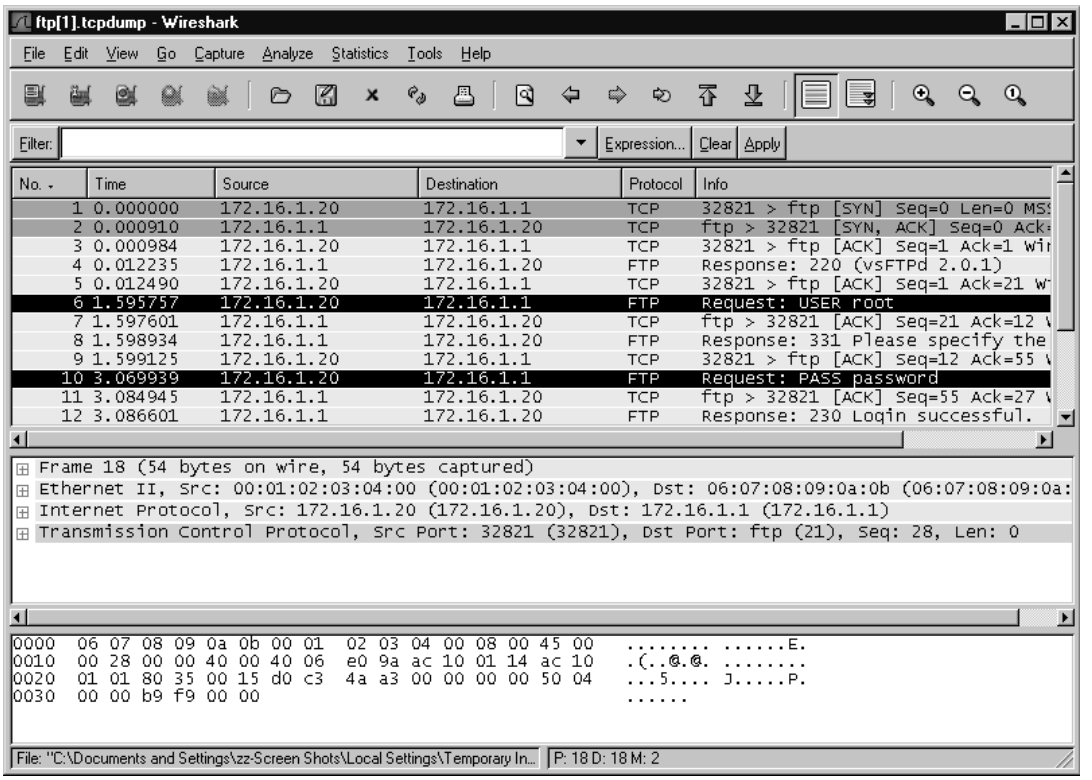
NOTE

A rootkit is a collection of Trojan programs that are used to replace the legitimate programs on a compromised system in order to avoid detection. Some common commands that are replaced are **ps**, **ifconfig**, and ***ls***. Rootkits can also install additional software such as sniffers.

What Does Sniffed Data Look Like?

The easiest way to grasp the concept of a sniffer is to watch one in action. Figure 7.2 shows a capture of a simple FTP session from a laptop to a Linux system. The two highlighted packets show how easy it is to sniff the username and password (i.e., "root" and "password").

Figure 7.2 Sniffing a Connection



Common Network Analyzers

A simple search on SecurityFocus (www.securityfocus.org/tools/category/4) shows the diversity and number of sniffers available. Some of the most prominent are:

- **Wireshark** Wireshark is one of the best sniffers available and is being developed as a free, commercial-quality sniffer. It has numerous features, a nice graphical user interface (GUI), decodes over 400 protocols, and is actively being developed and maintained. It runs on Uniplexed Information and Computing (UNIX)-based systems, Mac OS X, and Windows. This is a great sniffer to use in a production environment, and is available at www.wireshark.org.
- **WinDump** WinDump is the Windows version of tcpdump, and is available at www.winpcap.org/windump. It uses the WinPcap library and runs on Windows 95, 98, ME, NT, 2000, and XP.
- **Network General Sniffer** A Network General Sniffer is one of the most popular commercial sniffers available. Now a suite of enterprise network capture tools, there is an entire Sniffer product line at www.networkgeneral.com.

- **Windows 2000 and 2003 Server Network Monitor** Both the Windows 2000 Server and the Windows 2003 Server have a built-in program to perform network analysis. It is located in the "Administrative Tools" folder, but is not installed by default; therefore, you have to add it from the installation CD.
- **EtherPeek** EtherPeek is a commercial network analyzer developed by WildPackets. Versions for both Windows and Mac, and other network analysis products can be found at www.wildpackets.com.
- **Tcpdump** Tcpdump is the oldest and most commonly used network sniffer, and was developed by the Network Research Group (NRG) of the Information and Computing Sciences Division (ICSD) at Lawrence Berkeley National Laboratory (LBNL). It is command line-based and runs on UNIX-based systems, including Mac OS X. It is actively developed and maintained at www.tcpdump.org.
- **Snoop** Snoop is a command-line network sniffer that is included with the Sun Solaris OS.
- **Snort** Snort is a network IDS that uses network sniffing, and is actively developed and maintained at www.snort.org. For more information, refer to *Nessus, Snort, & Ethereal Power Tools: Customizing Open Source Security Applications* (Syngress Publishing: 1-597490-20-2) and *Snort Intrusion Detection and Prevention Toolkit* (Syngress, ISBN: 1597490997).
- **Dsniff** Dsniff is a very popular network-sniffing package. It is a collection of programs that are used to specifically sniff for interesting data (e.g., passwords) and to facilitate the sniffing process (e.g., evading switches). It is actively maintained at www.monkey.org/~dugsong/dsniff.
- **Ettercap** Ettercap was specifically designed to sniff a switched network. It has built-in features such as password collecting, OS fingerprinting, and character injection, and runs on several platforms including Linux, Windows, and Solaris. It is actively maintained at ettercap.sourceforge.net.
- **Analyzer** Analyzer is a free sniffer that is used for the Windows OS. It is being actively developed by the makers of WinPcap and WinDump at Politecnico di Torino, and can be downloaded from analyzer.polito.it.
- **Packetyzer** Packetyzer is a free sniffer (used for the Windows OS) that uses Wireshark's core logic. It tends to run a version or two behind the current release of Wireshark. It is actively maintained by Network Chemistry at www.network-chemistry.com/products/packetyzer.php.

- **MacSniffer** MacSniffer is specifically designed for the Mac OS X environment. It is built as a front-end for tcpdump. The software is shareware and can be downloaded from personalpages.tds.net/~brian_hill/macsniffer.html.

How Does It Work?

This section provides an overview of how sniffing takes place, and gives background information on how networks and protocols work. However, there are many other excellent resources available, including the most popular and undoubtedly one of the best written, Richard Stevens' "TCP/IP Illustrated, Vol. 1–3."

Explaining Ethernet

Ethernet is the most popular protocol standard used to enable computers to communicate. A protocol is like speaking a particular language. Ethernet was built around the principle of a shared medium where all computers on the local network segment share the same cable. It is known as a *broadcast* protocol because it sends that data to all other computers on the same network segment. This information is divided up into manageable chunks called *packets*, and each packet has a header containing the addresses of both the destination and source computers. Even though this information is sent out to all computers on a segment, only the computer with the matching destination address responds. All of the other computers on the network still see the packet, but if they are not the intended receiver they disregard it, unless a computer is running a sniffer. When running a sniffer, the packet capture driver puts the computer's NIC into *promiscuous mode*. This means that the sniffing computer can see all of the traffic on the segment regardless of who it is being sent to. Normally computers run in non-promiscuous mode, listening for information designated only for themselves. However, when a NIC is in promiscuous mode, it can see conversations to and from all of its neighbors.

Ethernet addresses are also known as Media Access Control (MAC) addresses and hardware addresses. Because many computers may share a single Ethernet segment, each one must have an individual identifier hard-coded onto the NIC. A MAC address is a 48-bit number, which is also stated as a 12-digit hexadecimal number. This number is broken down into two halves; the first 24 bits identify the vendor of the Ethernet card, and the second 24 bits comprise a serial number assigned by the vendor.

The following steps allow you to view your NIC's MAC address:

- **Windows 9x/ME** Access **Start | Run** and type **winipcfg.exe**. The MAC address will be listed as the "Adapter Address."
- **Windows NT, 2000, XP, and 2003** Access the command line and type **ipconfig /all**. The MAC address will be listed as the "Physical Address."

- **Linux and Solaris** Type **ifconfig –a** at the command line. The MAC address will be listed as the "HWaddr" on Linux and as "ether" on Solaris.
- **Macintosh OS X** Type **ifconfig –a** at the Terminal application. The MAC address will be listed as the "Ether" label.

You can also view the MAC addresses of other computers that you have recently communicated with, by typing the command **arp –a**. (Discussed in more detail in the "Defeating Switches" section.)

MAC addresses are unique, and no two computers have the same one. However, occasionally a manufacturing error may occur that causes more than one NIC to have the same MAC address. Thus, most people change their MAC addresses intentionally, which can be done with a program (e.g., *ifconfig*) that allows you to fake your MAC address. Faking your MAC address (and other types of addresses) is also known as *spoofing*. Also, some adapters allow you to use a program to reconfigure the runtime MAC address. And lastly, with the right tools and skill you can physically re-burn the address into the NIC.

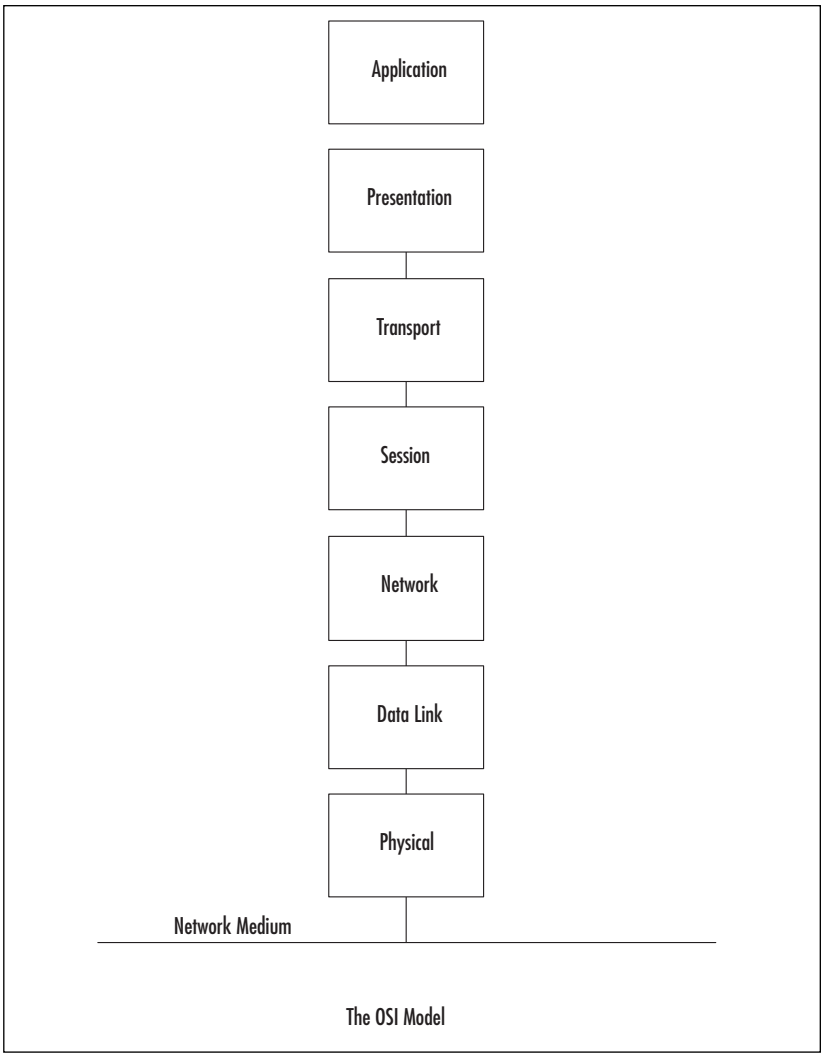
NOTE

Spoofing is the process of altering network packet information (e.g., the IP source address, the MAC address, or the e-mail address). This is often done to masquerade as another device in order to exploit a trust relationship or to make tracing the source of attacks difficult. Address spoofing is also used in DoS attacks (e.g., Smurf), where the return addresses of network requests are spoofed to be the IP address of the victim.

Understanding the Open Systems Interconnection Model

The International Standards Organization (ISO) developed the Open Systems Interconnection (OSI) model in the early 1980s to describe how network protocols and components work together. It divides network functions into seven layers, each layer representing a group of related specifications, functions, and activities (see Figure 7.3). Although complicated at first, the terminology is used extensively in networking, systems, and development communities.

Figure 7.3 Seven boxes corresponding to OSI model.



NOTE

The OSI model is not necessarily reflective of the way that applications and OSes are actually written. In fact, some security tools use the differences in protocol implementations to extract information from computers (including their OSes) and specific patches and services packs that may have been installed.

"*We still talk about the seven layers model, because it's a convenient model for discussion, but that has absolutely zero to do with any real-life software engineering. In other words, it's a way to talk about things, not to*

implement them. And that's important. Specs are a basis for talking about things. But they are not a basis for implementing software."

– Linus Torvalds, project coordinator for the Linux kernel, in an e-mail dated September 29, 2005.

The following sections define the seven layers of the OSI model.

Layer 1: Physical

The first layer of the OSI model is the *Physical* layer, which specifies the electrical and mechanical requirements for transmitting data bits across the transmission medium (cable or airwaves). It involves sending and receiving the data stream on the carrier, whether that carrier uses electrical (cable), light (fiber optic), radio, infrared, or laser (wireless) signals. The Physical layer specifications include:

- Voltage changes
- The timing of voltage changes
- Data rates
- Maximum transmission distances
- The physical connectors to the transmission medium (plug)
- The topology or physical layout of the network

Many complex issues are addressed at the Physical layer, including digital vs. analog signaling, baseband vs. broadband signaling, whether data is transmitted synchronously or asynchronously, and how signals are divided into channels (multiplexing).

Devices that operate at the Physical layer deal with signaling (e.g., transceivers on the NIC), repeaters, basic hubs, and simple connectors that join segments of cable). The data handled by the Physical layer is in bits of 1s (ones) and 0s (zeros), which are represented by pulses of light or voltage changes of electricity, and by the state of those pulses (*on* generally representing 1 and *off* generally representing *0*).

How these bits are arranged and managed is a function of the Data Link layer (layer 2) of the OSI model.

Layer 2: Data Link

Layer 2 is the *Data Link* layer, which is responsible for maintaining the data link between two computers, typically called *hosts* or *nodes*. It also defines and manages the ordering of bits to and from packets. *Frames* contain data arranged in an organized manner, which provides an orderly and consistent method of sending data bits across the medium. Without such

control, the data would be sent in random sizes or configurations and the data on one end could not be decoded at the other end. The Data Link layer manages the physical addressing and synchronization of the data packets. It is also responsible for flow control and error notification on the Physical layer. Flow control is the process of managing the timing of sending and receiving data so that it doesn't exceed the capacity (speed, memory, and so on) of the physical connection. Since the Physical layer is only responsible for physically moving the data onto and off of the network medium, the Data Link layer also receives and manages error messaging related to the physical delivery of packets.

Network devices that operate at this layer include layer 2 switches (switching hubs) and bridges. A layer 2 switch decreases network congestion by sending data out only on the port that the destination computer is attached to, instead of sending it out on all ports. Bridges provide a way to segment a network into two parts and filter traffic, by building tables that define which computers are located on which side of the bridge, based on their MAC addresses.

The Data Link layer is divided into two sublayers: the Logical Link Control (LLC) sublayer and the MAC sublayer.

The MAC Sublayer

The MAC sublayer provides control for accessing the transmission medium. It is responsible for moving data packets from one NIC to another, across a shared transmission medium such as an Ethernet or fiber-optic cable.

Physical addressing is addressed at the MAC sublayer. Every NIC has a unique MAC address (also called the *physical address*) which identifies that specific NIC on the network. The MAC address of a NIC is usually burned into a read-only memory (ROM) chip on the NIC. Each manufacturer of network cards is provided a unique set of MAC addresses so that theoretically, every NIC that is manufactured has a unique MAC address. To avoid any confusion, MAC addresses are permanently burned into the NIC's memory, which is sometimes referred to as the Burned-in Address (BIA).

NOTE

On Ethernet NICs, the physical or MAC address (also called the *hardware* address) is expressed as 12 hexadecimal digits arranged in pairs with colons between each pair (e.g., 12:3A:4D:66:3A:1C). The initial three sets of numbers represent the manufacturer, and the last three bits represent a unique NIC made by that manufacturer.

MAC refers to the method used to allocate network access to computers while preventing them from transmitting at the same time and causing data collisions. Common

MAC methods include Carrier Sense Multiple Access/Collision Detection (CSMA/CD) used by Ethernet networks, Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) used by AppleTalk networks, and token passing used by Token Ring and Fiber Distributed Data Interface (FDDI) networks. (CSMA/CD is discussed later in this chapter.)

The LLC Sublayer

The LLC sublayer provides the logic for the data link; thus it controls the synchronization, flow control, and error-checking functions of the Data Link layer. This layer manages connection-oriented transmissions; however, connectionless service can also be provided by this layer. Connectionless operations are known as Class I LLC, whereas Class II can handle either connectionless or connection-oriented operations. With connection-oriented communication, each LLC frame sent is acknowledged. The LLC sublayer at the receiving end keeps up with the LLC frames it receives (also called Protocol Data Units [PDUs]); therefore, if it detects that a frame has been lost during transmission, it can send a request to the sending computer to start the transmission over again, beginning with the PDU that never arrived.

The LLC sublayer sits above the MAC sublayer, and acts as a liaison between the upper layers and the protocols that operate at the MAC sublayer (e.g., Ethernet, Token Ring, and so on). The LLC sublayer is defined by Institute of Electrical & Electronics Engineers (IEEE) 802.2. Link addressing, sequencing, and definition of Service Access Points (SAPs) also take place at this layer.

Layer 3: Network

The next layer is the *Network* layer (layer 3), which is where packets are sequenced and logical addressing is assigned. Logical addresses are nonpermanent, software-assigned addresses that can only be changed by administrators. The IP addresses used by the TCP/IP protocols on the Internet, and the Internet Package Exchange (IPX) addresses used by the IPX/Sequenced Packet Exchange (SPX) protocols on NetWare networks are examples of logical addresses. These protocol stacks are referred to as *routable* because they include addressing schemes that identify the network or subnet and the particular client on that network or subnet. Other network/transport protocols (e.g., NETBIOS Extended User Interface [NetBEUI]) do not have a sophisticated addressing scheme and thus cannot be routed between different types of networks.

NOTE

To understand the difference between physical and logical addresses, consider this analogy: A house has a physical address that identifies exactly where it is located. This is similar to the MAC address on a NIC.

A house also has a logical address assigned to it by the post office that consists of a street name and number. The post office occasionally changes

the names of streets or renumbers the houses located on them. This is similar to the IP address assigned to a network interface.

The Network layer is also responsible for creating a virtual circuit (i.e., a logical connection, not a physical connection) between points or nodes. A node is a device that has a MAC address, which typically includes computers, printers, and routers. This layer is also responsible for routing, layer 3 switching, and forwarding packets. *Routing* refers to forwarding packets from one network or subnet to another. Without routing, computers can only communicate with computers on the same network. Routing is the key to the global Internet, and is one of the most important duties of the Network layer.

Finally, the Network layer provides additional levels of flow control and error control. As mentioned earlier, from this point on, the primary methods of implementing the OSI model architecture involve software rather than hardware.

Devices that operate at this layer include routers and layer 3 switches.

Layer 4: Transport

Layer 4 is the *Transport* layer, and is responsible for transporting the data from one node to another. It provides transparent data transfer between nodes, and manages the end-to-end flow control, error detection, and error recovery.

The Transport layer protocols initiate contact between specific ports on different host computers, and set up a virtual circuit. The transport protocols on each host computer verify that the application sending the data is authorized to access the network and that both ends are ready to initiate the data transfer. When this synchronization is complete, the data is sent. As the data is being transmitted, the transport protocol on each host monitors the data flow and watches for transport errors. If transport errors are detected, the transport protocol provides error recovery.

The functions performed by the Transport layer are very important to network communication. Just as the Data Link layer provides lower-level reliability and connection-oriented or connectionless communications, the Transport layer does the same thing but at a higher level. The two protocols most commonly associated with the Transport layer are the TCP, which is connection-oriented and the User Datagram Protocol (UDP), which is connectionless.

NOTE

What's the difference between a connection-oriented protocol and a connectionless protocol? A connection-oriented protocol (e.g., TCP) creates a connection between two computers before sending the data, and then verifies that the data has reached its destination by using acknowledgements (ACKs) (i.e., messages sent back to the sending computer from the receiving computer that acknowledge receipt). Connectionless protocols send the data and trust that it will reach the proper destination or that the application will handle retransmission and data verification.

Consider this analogy: You need to send an important letter to a business associate that contains valuable papers. You call him before e-mailing the letter, to let him know that he or she should expect it (establishing the connection). A few days later your friend calls to let you know that he received the letter, or you receive the return receipt (ACK). This is how connection-oriented communication works. When mailing a postcard to a friend, you drop it in the mailbox and hope it gets to the addressee. You don't expect or require any acknowledgement. This is how connectionless communication works.

The Transport layer also manages the logical addressing of ports. Think of a port as a suite or apartment number within a building that defines exactly where the data should go.

Table 7.1 Commonly Used Internet Ports

Internet Protocol (IP) Port(s)	Protocol(s)	Description
80	TCP	HTTP, commonly used for Web servers
443	TCP	Hypertext Transfer Protocol Secure sockets (HTTPS) for secure Web servers.
53	UDP and TCP	Domain Name Server/Service (DNS) for resolving names to IP addresses
25	TCP	SMTP, used for sending e-mail
22	TCP	The Secure Shell (SSH) protocol
23	TCP	Telnet, an insecure administration protocol
20 and 21	TCP	An insecure FTP

Continued

Table 7.1 continued Commonly Used Internet Ports

Internet Protocol (IP) Port(s)	Protocol(s)	Description
135–139 and 445	TCP and UDP	Windows file sharing, login, and Remote Procedure Call (RPC)
500	UDP	Internet Security Association and Key Management Protocol (ISAKMP) key negotiation for Secure Internet Protocol (IPSec) virtual private networks (VPNs)
5060	UDP	Session Initiation Protocol (SIP) for some VoIP uses
123	UDP	Network Time Protocol (NTP) for network time synchronization

A computer may have several network applications running at the same time (e.g., a Web browser sending a request to a Web server for a Web page, an e-mail client sending and receiving e-mail, and a file transfer program uploading or downloading information to and from an FTP server). The mechanism for determining which incoming data packets belong to which application is the function of port numbers. The FTP protocol is assigned a particular port, whereas the Web browser and e-mail clients use different protocols (e.g., HTTP and POP3 or Internet Message Access Protocol [IMAP]) that have their own assigned ports; thus the information intended for the Web browser doesn't go to the e-mail program by mistake. Port numbers are used by TCP and UDP.

Finally, the Transport layer deals with name resolution. Most users prefer to identify computers by name instead of by IP address (i.e., *www.microsoft.com* instead of 207.46.249.222), however, computers only interpret numbers, therefore, there must be a way to match names with numerical addresses. Name resolution methods such as the DNS solve this problem.

Layer 5: Session

After the Transport layer establishes a virtual connection, a communication session is made between two processes on two different computers. The Session layer (layer 5) is responsible for establishing, monitoring, and terminating sessions, using the virtual circuits established by the Transport layer.

The Session layer is also responsible for putting header information into data packets that indicates where a message begins and ends. Once header information is attached to the data packets, the Session layer performs synchronization between the sender's Session layer and

the receiver's Session layer. The use of ACKs helps coordinate the transfer of data at the Session-layer level.

Another important function of the Session layer is controlling whether the communications within a session are sent as full-duplex or half-duplex messages. Half-duplex communication goes in both directions between the communicating computers, but information can only travel in one direction at a time (e.g., radio communications where you hold down the microphone button to transmit, but cannot hear the person on the other end). With full-duplex communication, information can be sent in both directions at the same time (e.g., a telephone conversation, where both parties can talk and hear one another at the same time).

Whereas the Transport layer establishes a connection between two machines, the Session layer establishes a connection between two processes. An application can run many processes simultaneously to accomplish the work of the application.

After the Transport layer establishes the connection between the two machines, the Session layer sets up the connection between the application process on one computer and the application process on another computer.

Layer 6: Presentation

Data translation is the primary activity of the Presentation layer (layer 6). When data is sent from a sender to a receiver, it is translated at the Presentation layer (i.e., the sender's application passes data down to the Presentation layer, where it is changed into a common format). When the data is received on the other end, the Presentation layer changes it from the common format back into a format that is useable by the application. Protocol translation (i.e., the conversion of data from one protocol to another so that it can be exchanged between computers using different platforms or OSes) takes place here.

The Presentation layer is also where *gateway* services operate. Gateways are connection points between networks that use different platforms or applications (e.g., e-mail gateways, Systems Network Architecture (SNA) gateways, and gateways that cross platforms or file systems). Gateways are usually implemented via software such as the Gateway Services for NetWare (GSNW). Software redirectors also operate at this layer.

This layer is also where data compression takes place, which minimizes the number of bits that must be transmitted on the network media to the receiver. Data encryption and decryption also take place in the Presentation layer.

Layer 7 Application

The *Application* layer is the point at which the user application program interacts with the network. Don't confuse the networking model with the application itself. Application processes (e.g., file transfers or e-mail) are initiated within a user application (e.g., an e-mail program). Then the data created by that process is handed to the Application layer of the networking software. Everything that occurs at this level is application-specific (e.g., file

sharing, remote printer access, network monitoring and management, remote procedure calls, and all forms of electronic messaging).

Both FTP and Telnet function within the Application layer, as does the SMTP, Post Office Protocol (POP), and Internet IMAP, all of which are used for sending or receiving e-mail. Other Application-layer protocols include HTTP, Network News Transfer Protocol (NNTP), and SNMP.

You have to distinguish between the protocols mentioned and the applications that might bear the same names, because there are many different FTP programs made by different software vendors that use the FTP to transfer files.

The OSI model is generic and can be used to explain all network protocols. Various protocol suites are often mapped against the OSI model for this purpose. A solid understanding of the OSI model aids in network analysis, comparison, and troubleshooting. However, it is important to remember that not all protocols map well to the OSI model (e.g., TCP/IP was designed to map to the U.S. Department of Defense (DoD) model). In the 1970s, the DoD developed its four-layer model. The core Internet protocols adhere to this model.

The DoD model is a condensed version of the OSI model. Its four layers are:

- **Process Layer** This layer defines protocols that implement user-level applications (e.g., e-mail delivery, remote login, and file transfer.
- **Host-to-host Layer** This layer manages the connection, data flow management, and retransmission of lost data.
- **Internet Layer** This layer delivers data from the source host to the destination host across a set of physical networks that connect the two machines.
- **Network Access Layer** This layer manages the delivery of data over a particular hardware media.

Notes From the Underground…

Writing Your Own Sniffer

There is an excellent paper titled "Basic Packet-Sniffer Construction from the Ground Up" by Chad Renfro that is located at www.packetstormsecurity.org/sniffers/Sniffer_construction.txt. In this paper, he presents a basic 28-line packet sniffer that is written in C, called *sniff.c*, which he explains line-by-line in an easy-to-understand manner. The program demonstrates how to use the *raw_socket* device to read TCP packets from the network, and how to print basic header information to *stdout*.

Continued

For simplicity, the program operates in non-promiscuous mode; therefore, you first need to put your interface in promiscuous mode using the **ifconfig eth0 promisc** command.

There is also a header file that must be copied into the same directory as *sniff.c*, that provides standard structures to access the IP and TCP fields in order to identify each field in the IP and TCP header.

To run the program, copy the *sniff.c* and *headers.h* files into one directory, and enter **gcc -o sniff sniff.c**. This compiles the program and creates an executable file called *sniff*, which is run by typing **./sniff**. The following text shows the output of the sniff program when a Telnet and FTP connection was attempted:

```
Bytes received :::    48
Source address ::: 192.168.1.1
IP header length ::: 5
Protocol ::: 6
Source port ::: 1372
Dest port  ::: 23
Bytes received :::    48
Source address ::: 192.168.1.1
IP header length ::: 5
Protocol ::: 6
Source port ::: 1374
Dest port  ::: 21
```

Once you are done capturing data, you can end the program by typing **Ctrl-C**. You may also want to remove your interface from promiscuous mode by typing the **ifconfig eth0 –promisc** command.

CSMA/CD

Ethernet uses the CSMA/CD protocol in order for devices to exchange data on the network. The term *multiple access* refers to the fact that many network devices attached to the same segment have the opportunity to transmit. Each device is given an equal opportunity; no device has priority over another. *Carrier sense* describes how an Ethernet interface on a network device listens to the cable before transmitting. The network interface ensures that there are no other signals on the cable before it transmits, and listens while transmitting to ensure that no other network device transmits data at the same time. When two network devices transmit at the same time, a *collision* occurs. Because Ethernet interfaces listen to the media while they are transmitting, they can identify the presence of others through *collision detection*. If a collision occurs, the transmitting device waits for a small, random amount of time before retransmitting. This function is known as *random backoff*.

Traditionally, Ethernet operation has been half-duplex, which means that an interface can either transmit or receive data, but not at the same time. If more than one network interface on a segment tries to transmit at the same time, a collision occurs per CSMA/CD. When a crossover cable is used to connect two devices, or a single device is attached to a switch port, only two interfaces on the segment need to transmit or receive; no collisions occur. This is because the transmit (TX) of device A is connected to the receive (RX) of device B, and the TX of B is connected to the RX of device A. The collision detection method is no longer necessary, therefore, interfaces can be placed in full-duplex mode, which allows network devices to transmit and receive at the same time, thereby increasing performance.

The Major Protocols: IP, TCP, UDP, and ICMP

The next four protocols are at the heart of how the Internet works today.

NOTE

Other, different protocols are used across the Internet, and new protocols are constantly created to fulfill specific needs. One of these is Internet Protocol version 6 (IPv6), which seeks to improve the existing Internet protocol suite by providing more IP addresses, and by improving the security of network connections across the Internet using encryption. For more information on IPv6, see http://en.wikipedia.org/wiki/IPv6.

IP

The IP is a connectionless protocol that manages addressing data from one point to another, and fragments large amounts of data into smaller, transmittable packets. The major components of Internet Protocol datagrams are:

- **IP Identification (IPID)** Tries to uniquely identify an IP datagram.
- **Protocol** Describes the higher-level protocol contained within the datagram.
- **Time-to-live (TTL)** Attempts to keep datagrams and packets from routing in circles. When TTL reaches 0, the datagram is dropped. The TTL allows traceroute to function, identifying each router in a network by sending out datagrams with successively increasing TTLs, and tracking when those TTLs are exceeded.
- **Source IP Address** The IP address of the host where the datagram was created.
- **Destination IP Address** The destination of where the datagram should be sent.

Notes from the Underground…

IP Address Source Spoofing

It is possible to spoof any part of an IP datagram; however, the most commonly spoofed IP component is the source IP address. Also, not all protocols function completely with a spoofed source IP address (e.g., connection-oriented protocols such as TCP require handshaking before data can be transmitted, thereby reducing the effectiveness of spoofing-based attacks).

Spoofing can also be used as a DoS attack. If Network A sends a datagram to Network B, with a spoofed source IP host address on Network C, Network C will see traffic going to it that originates from Network B, perhaps without any indication that Network A is involved at all.

The best practice for network administrators is to ensure that the network can only originate packets with a proper Source IP address (i.e., an IP address in the network itself).

Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) manages errors that occur between networks on the IP. The following are common types of ICMP messages:

- **Echo Request/Reply** Used by programs such as *ping* to calculate the delay in reaching another IP address.
- **Destination Unreachable: Network Unreachable and Port Unreachable** Sent to the source IP address of a packet when a network or port cannot be reached. This happens when a firewall rejects a packet or if there is a network problem. There are a number of subtypes of Destination Unreachable messages that are helpful at diagnosing communication issues.
- **Time Exceeded** Occurs when a packet reaches 0.

TCP

TCP packets are connection-oriented, and are used most often to transmit data. The connection-oriented nature of TCP packets makes it a poor choice for source IP address spoofing. Many applications use TCP, including the Web (HTTP), e-mail (SMTP), FTP, SSH, and many Windows FTPs.

The TCP Handshake

An important concept of the TCP is *handshaking*. Before any data can be exchanged between two hosts, they must agree to communicate. Host A sends a packet with the SYN flag set to Host B. If Host B is willing and able to communicate, it returns the SYN packet and adds an ACK flag. Host A begins sending data, and indicates to Host B that it also received the ACK. When the communication between the hosts ends, a packet with the FIN (finish) flag is sent, and a similar acknowledgement process is followed.

TCP Sequence

Another important component of TCP is *sequence identification*, where each packet sent is part of a sequence. Through these numbers, TCP handles complex tasks such as retransmission, acknowledgement, and order.

UDP

UDP packets are the connectionless equivalent to TCP, and are used for many purposes, the most important being that DNS uses UDP for most of its work. DNS finds out which IP address corresponds to which hostname (e.g., www.example.com is not routable as an IP address inside an IP datagram; however, through a DNS system it can find the IP address to route traffic to). Other uses of UDP include VoiP and many online games and streaming media types.

Hardware: Cable Taps, Hubs, and Switches

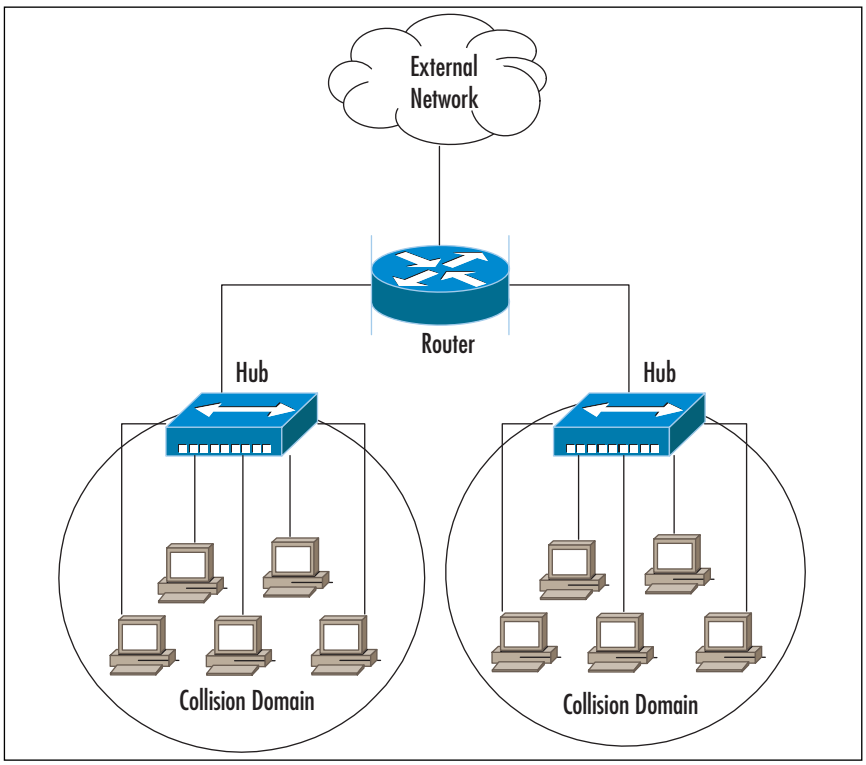
Cable taps are hardware devices that assist in connecting to a network cable. Test access points (Taps) use this device to access any cables between computers, hubs, switches, routers, and other devices. Taps are available in full- or half-duplex for 10, 100, and 1,000 Mbps Ethernet links. They are also available in various multi-port sizes. The following is a list of some popular cable tap products:

- Net Optics carries several types of network taps for copper and fiber cables, and is available at www.netoptics.com.
- The Finisar Tap family offers a variety of taps for copper and fiber cables, and is available at www.finisar.com/nt/taps.php.

A *hub* is a device that allows you to connect multiple hosts together on a shared medium (e.g., Ethernet). When a computer sends information, it travels into the hub and the hub forwards the information to all other computers connected to it. The computer that the information was intended for will recognize its own MAC address in the packet header and accept the data. The area that the hub forwards all information to is known as a *collision domain* (also known as *broadcast domain*). A hub has only one collision domain for all traffic to

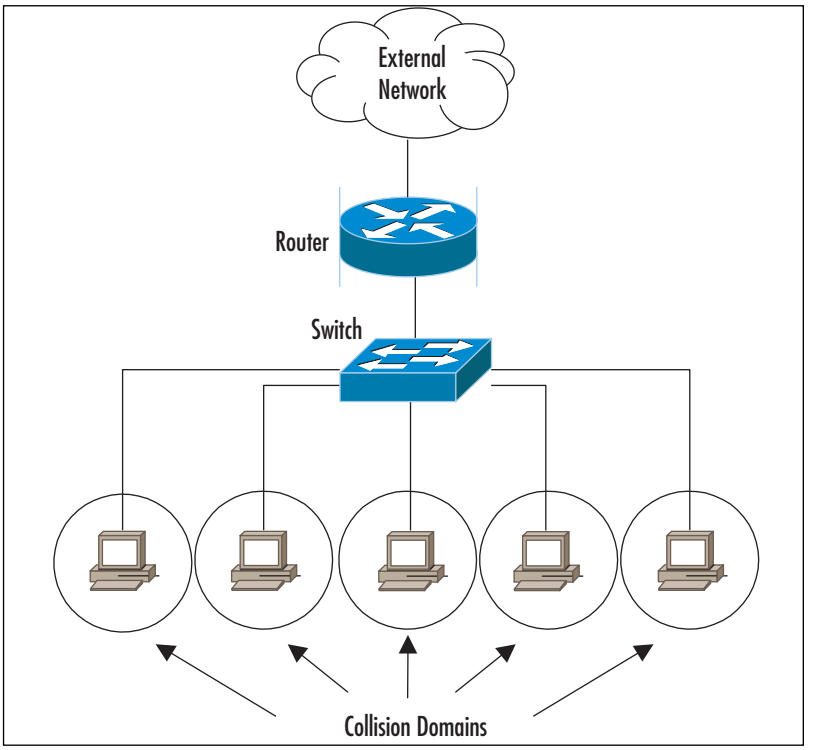
share. Figure 7.4 shows a network architecture with collision domains related to hubs. Large collisions make sniffing easier and creates performance issues such as bandwidth hogging or excessive traffic on the hub.

Figure 7.4 Hub Collision Domains



A switch is also used to connect computers together on a shared medium; however, when a switch receives information, it doesn't blindly send it to all other computers; it looks at the packet header to locate the destination MAC address, and maintains a list of all MAC addresses and corresponding ports on the switch that the computers are connected to. It then forwards the packets to the specified port. This narrows the collision domain to a single port (see Figure 7.5). This type of collision domain also provides a definite amount of bandwidth for each connection rather than a shared amount on a hub. Because the price of switches has fallen dramatically in the last few years, there is no reason not to replace hubs with switches, or to choose switches when purchasing new equipment. Also, some of the more costly switches include better technology that makes them more resistant to sniffing attacks.

Figure 7.5 Switch Collision Domains



As you can see from the diagrams, hubs make sniffing easier and switches make sniffing more difficult. However, switches can be tricked, as discussed in the "Defeating Switches" section.

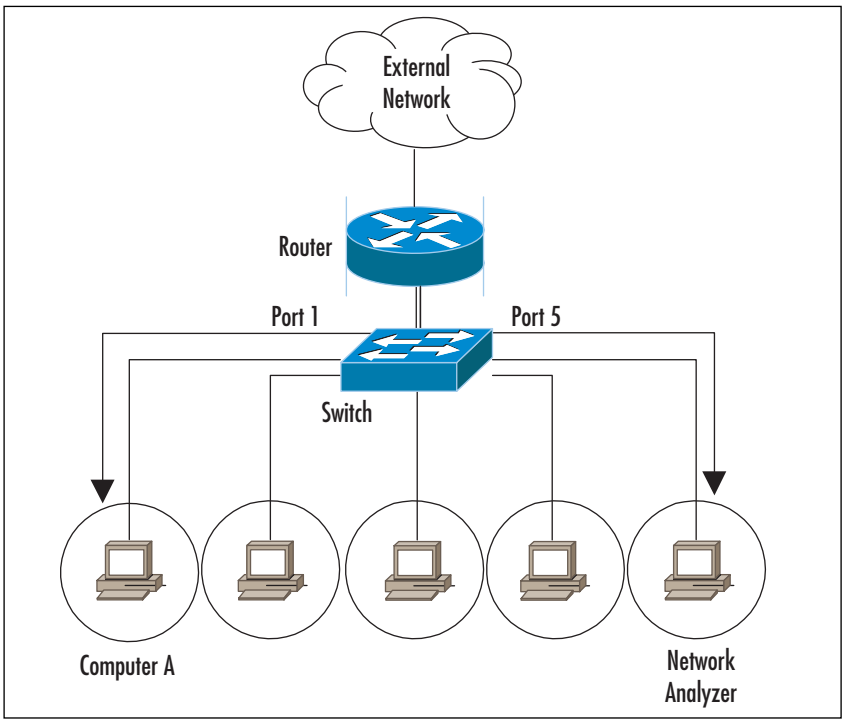
Port Mirroring

If you are working on a network that uses switches and you want to perform network analysis legitimately, you are in luck; most switches and routers come with *port mirroring* (also known as *port spanning*). To mirror ports, you have to configure the switch to duplicate the traffic from the port you want to monitor to the port you are connected to.

Using port spanning does not interfere with the normal operation of switches, but you should always check the documentation of the exact switch you are configuring and periodically check the device's logs. You won't affect the switch, but you will increase the amount of traffic on a specific destination port, therefore, make sure your properly configured network analyzer is the destination port. Also, consult the documentation for your specific switch to learn the exact command to enable port mirroring (see Figure 7.6). The switch is configured to mirror all port 1 traffic to port 5, and the network analyzer sees all traffic to

and from Computer A. Sometimes administrators mirror the uplink port on a switch, so that they can see all of the traffic to and from the switch and all of its ports.

Figure 7.6 Port Mirroring



NOTE

Span stands for Switched Port Analyzer. Cisco uses the word *span* to describe the concept of port mirroring. In Cisco terms, spanning a port is the same as mirroring a port.

Defeating Switches

As mentioned earlier, using switches on a network makes sniffing more difficult. In theory, you should only see traffic destined for you own computer on a switch; however, there are ways to circumvent its technology. The following list describes several ways in which a switch can be defeated:

- **Switch Flooding** Some switches can be made to act like a hub, where all packets are broadcast to all computers. This can be accomplished by overflowing the switch

address table with a large number of fake MAC addresses (known as a device *failing open)*, thus removing all security provisions. Devices that *fail close* incorporate some type of security measure (e.g., shutting down all communications). The Dsniff package comes with a program called *macof* that is designed to switch MAC address flooding. It can be downloaded from www.monkey.org/~dugsong/dsniff.

- **ARP Redirects** When a computer needs the MAC address of another computer, it sends an Address Resolution Protocol (ARP) request. Each computer also maintains an ARP table that stores the MAC addresses of the computers it talks to. ARPs are broadcast on a switch; therefore, all computers on that switch see the request and the response. There are several methods that use ARP to trick a switch into sending traffic somewhere it shouldn't. First, an intruder can subvert a switch by sending out an ARP claiming to be someone else. An intruder can also send an ARP claiming to be the router, in which case computers will try to send their packets through the intruder's computer. Or, an intruder will send an ARP request to just one victim, claiming to be the router, at which point the victim starts forwarding packets to the intruder.
- **ICMP Redirect** Sometimes computers are on the same physical segment and switch, but different logical segments. This means they are in different IP subnets. When Computer A wants to talk to Computer B it sends its request through a router. The router knows that they are on the same physical segment, so it sends an ICMP Redirect to Computer A letting it know that it can send its packets directly to Computer B. An intruder (Computer X) can send a fake ICMP redirect to Computer A, telling it to send Computer B's packets to Computer X.
- **ICMP Router Advertisements** These advertisements tell computers which router to use. Intruders then send out advertisements claiming to be that router, at which point the computers begin forwarding all packets through the intruder.
- **MAC Address Spoofing** An intruder can pretend to use a different computer by spoofing its MAC address. Sending out packets with the source address of the victim tricks the switch. The switch enters the spoofed information into its table and begins sending packets to the intruder. But what about the victim who is still on the switch, sending updates that are causing the switch to change the table back? This can be solved by taking the victim offline with some type of DoS attack, and then redirecting the switch and continuing communications. An intruder could also broadcast the traffic that he or she receives to ensure that the victim computer still receives the packets. Some switches have a countermeasure that allows you to statically assign a MAC address to a port. This may be difficult to manage if you have a large network, but it will eliminate MAC spoofing. Other switch configurations allow a port to be locked to the first MAC it encounters, and

presents a compelling balance between manageability and security in environments where physical port access is restricted.

To spoof your MAC on Linux or Solaris use the **ifconfig** command as follows:

```
ifconfig eth0 down
ifconfig eth0 hw ether 00:02:b3:00:00:AA
ifconfig eth0 up
```

Register the MAC on all hosts by broadcast ping: **ping -c 1 -b 192.168.1.255**.

Now you can sniff all traffic to the computer that owns this MAC address.

- **Reconfigure Port Spanning On the Switch** As mentioned earlier, switch ports can be configured to see traffic destined for other ports. An intruder can perform this by connecting to the switch via Telnet or some other default back door. An intruder can also use SNMP if it is not secured.
- **Cable Taps** As mentioned earlier, cable taps can be used to physically tap into the cable. Tapping into the uplink cable on a switch shows you all of the traffic entering and exiting that switch.

There are many methods for defeating switches that are contingent on how a switch operates. Not all of the methods discussed work, especially with new, more technologically savvy switches. The Dsniff Frequently Asked Questions (FAQ) helpful information for sniffing in a switched environment, and can be located at www.monkey.org/~dugsong/dsniff/faq.html.

Sniffing Wireless

From the airport, to the coffee shop, to the library, to your next door neighbor, wireless networks are all around us; therefore, wireless security is a serious concern. There are historical weaknesses in security protocols, because intruders no longer need to be inside a building to attack an internal network. A wireless network is still a network, however, and with a few exceptions maps well to the Ethernet and OSI models.

Hardware Requirements

While most Ethernet cards are capable of packet sniffing in promiscuous mode, many wireless chipsets cannot use *monitor* mode, which is the wireless equivalent of promiscuous mode. Complicating the situation is that wireless card manufacturers do not generally list the chipset that they use in a readily available form. Also, chipsets can vary within model families. It is best to select the software you want to use, and then identify which chipsets and specific manufacturer's model numbers work best with the specific drivers necessary for the software to function.

Here are some general guidelines on chipset compatibility:

- **Atheros** This chipset is compatible with most software and widely available in a number of adapters.
- **Prism2** This chipset is one of the most capable used with the Host AP drivers. Not only is it supported by most software, it can also run in an AP mode.
- **Orinoco** One of the first chipsets that supported monitor mode. Supported by most software. Cannot receive 802.11g traffic.
- **Broadcom** There is no native support in Linux for this chipset. With included drivers, tools such as Kismet do not function with it. You may be able to use Windows drivers through a Network Driver Interface Specification (NDIS) compatibility wrapper such as the commercial DriverLoader, which can be downloaded from www.linuxant.com/driverloader.

Software

The proper combination of hardware, software, and drivers will enable you to effectively sniff wireless networking traffic. The following tools may be helpful:

- **Netstumbler** Netstumbler is more of a network scanner than a network sniffing tool, but is useful for listing networks detectable from your location. Netstumbler is an active network scanner that sends out probes that are detectable by others. It can be downloaded for free from www.netstumbler.org.
- **Kismet** Kismet is an open-source, free, wireless network scanner and vulnerability detector, that keeps track of wireless clients and their network associations. Unlike other scanners, it is a completely passive network scanner, and can be downloaded from www.kismetwireless.net.
- **Wireshark** Wireshark has a number of dissectors for wireless management traffic; however, it does not track by Service Set Identifier (SSID), nor does it show signal strength.
- **CommView for WiFi** CommView for WiFi is a commercial wireless network monitor and scanner that can export in tcpdump format, which Wireshark imports and reads easily. CommView for WiFi can be downloaded from www.tamos.com/products/commwifi/.

NOTE

Bootable Compact Disk - Read-only Memory (CD-ROMs)

There are several bootable Linux distributions that come prepackaged with the correct drivers and software necessary for wireless and wired network sniffing. All of these include Kismet and Ethereal or Wireshark. Below are some that are available and free:

- **Backtrack** Backtrack is the result of two highly respected bootable penetration toolsets combining their efforts toward one unified bootable CDROM. For additional information, go to www.remote-exploit.org.
- **Professional Hacker's Linux Assault Kit (Phlack)** Includes many security tools and wireless auditing and scanning software. For additional information, go to www.phlak.org.
- **Knoppix Security Tools Distribution (Knoppix-STD)** A general-purpose collection of security tools on a bootable Linux image. For additional information, go to www.s-t-d.org.

Protocol Dissection

Now that we've reviewed many of the critical portions of layers 1 through 4 of the OSI networking model, some attention should be paid to some of the protocols that you may run across while using Wireshark. The larger the network that you are sniffing, the more types of protocols (and protocol anomalies) you are likely to encounter.

DNS

The DNS translates hostnames into IP addresses, and vice versa. Most DNS traffic is transferred over UDP port 53 in a client/server fashion. DNS can be considered *forward* or *reverse*. Forward DNS translates a hostname into an IP address, and reverse DNS translates an IP address into a hostname. On the protocol level, forward and reverse lookups are nearly identical.

To get an IP address from a given hostname, a DNS system (also known as a *resolver*) requests an address (A) record from a DNS server. In the following example, we ask the authoritative name server for the IP address of *www.example.com*. In tcpdump format, we see the following traffic:

```
IP 192.168.0.1.33141 > 192.0.34.43.53:  42827+ A? www.example.com.
IP 192.0.34.43.53 > 192.168.0.1.33141:  42827*- 1/2/2 A 192.0.34.166
```

The resolver (192.168.0.1) asked the authoritative name server (192.0.34.43) on UDP port 53 for the "A record" for www.example.com. Via UDP, the name server returned one A record for that name with IP address 192.0.34.166.

Wireshark can also be used to view more information about this DNS transaction. Wireshark would return the following information about the query and response:

```
Domain Name System (query)
    Transaction ID: 42827
    Flags: 0x0100 (Standard query)
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...0 .... = Non-authenticated data OK
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    Queries
        www.example.com: type A, class IN
            Name: www.example.com
            Type: A (Host address)
            Class: IN (0x0001)
Domain Name System (response)
    Transaction ID: 42827
    Flags: 0x8180 (Standard query response, No error)
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .0.. .... .... = Authoritative
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... 1... .... = Recursion available
        .... .... .0.. .... = Z: reserved (0)
        .... .... ..0. .... = Answer authenticated
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 13
    Additional RRs: 2
    Queries
```

```
        www.example.com: type A, class IN
            Name: www.example.com
            Type: A (Host address)
            Class: IN (0x0001)
    Answers
        www.example.com: type A, class IN, addr 192.0.34.166
    Authoritative nameservers
        com: type NS, class IN, ns C.GTLD-SERVERS.NET
       ...
        com: type NS, class IN, ns B.GTLD-SERVERS.NET
    Additional records
        A.GTLD-SERVERS.NET: type A, class IN, addr 192.5.6.30
```

NOTE

DNS uses TCP instead of UDP for transmitting data when the data size exceeds 512 bytes. DNS also uses TCP for transferring entire DNS zones between zones. In either case, port 53 is used.

NTP

The NTP is another helpful protocol that keeps things running smoothly in the background. In this case, NTP makes sure that all of your computer and device clocks are synchronized. NTP can use peering or client/server architecture; the network traffic will be similar either way. UDP port 123 is used for NTP.

In the following example, NTP client (192.168.0.1) asks a NTP server (192.168.0.2) for the current timestamp:

```
IP 192.168.0.1.ntp > 192.168.0.2.ntp: NTPv4, Client, length 48
IP 192.168.0.2.ntp > 192.168.0.1.ntp: NTPv4, Server, length 48

Network Time Protocol
    Flags: 0xe3
        11.. .... = Leap Indicator: alarm condition (clock not synchronized)
        ..10 0... = Version number: NTP Version 4 (4)
        .... .011 = Mode: client (3)
    Peer Clock Stratum: unspecified or unavailable (0)
    Peer Polling Interval: 6 (64 sec)
    Peer Clock Precision: 0.000008 sec
```

```
    Root Delay:    0.0000 sec
    Clock Dispersion:    0.0039 sec
    Reference Clock ID: Unindentified reference source 'INIT'
    Reference Clock Update Time: NULL
    Originate Time Stamp: Mar 29, 2006 06:09:01.6976 UTC
    Receive Time Stamp: Mar 29, 2006 06:09:01.7563 UTC
    Transmit Time Stamp: Mar 29, 2006 06:10:07.7525 UTC

Network Time Protocol
    Flags: 0x24
        00.. .... = Leap Indicator: no warning (0)
        ..10 0... = Version number: NTP Version 4 (4)
        .... .100 = Mode: server (4)
    Peer Clock Stratum: secondary reference (5)
    Peer Polling Interval: 6 (64 sec)
    Peer Clock Precision: 0.000008 sec
    Root Delay:    0.0000 sec
    Clock Dispersion:    0.0122 sec
    Reference Clock ID: 127.127.1.0
    Reference Clock Update Time: Mar 29, 2006 06:09:48.4681 UTC
    Originate Time Stamp: Mar 29, 2006 06:10:07.7525 UTC
    Receive Time Stamp: Mar 29, 2006 06:10:07.6674 UTC
    Transmit Time Stamp: Mar 29, 2006 06:10:07.6675 UTC
```

HTTP

HTTP is the most widely used protocol that supports the Web. HTTP uses TCP to transmit data exclusively, and in a default configuration uses port 80. Each object (e.g., Web page, image, audio) fetched from a Web server is transmitted via an individual HTTP session.

To begin an HTTP session, a client establishes a regular TCP connection on port 80 and sends a packet with the SYN flag set. A packet is returned from the Web server, with an ACK flag added to the SYN flag. Finally, the client sends a packet with the ACK flag set, and then sends another packet requesting a specific HTTP object.

The following is an example of a client's request to a HTTP server:

```
GET /index.html HTTP/1.1
```

The client requests the *index.html* page using HTTP v1.1:

```
Host: www.example.com
```

The hostname that was typed in the browser allows a server to host multiple Web services on one IP address:

```
User-Agent: ELinks/0.11.0 (textmode; Linux; 80x25-2)
```

The User-Agent describes the Web browser version to the server. Some browsers allow users to change hostnames; thereby deeming it unreliable.

```
Accept-Encoding: gzip
Accept-Language: en
Connection: Keep-Alive
```

These lines tell the Web server that the client supports compression of the object requested, accepts pages in English, and the Web server doesn't have to disconnect upon completion of the object request.

The Web server sends back the following information to the client:

```
HTTP/1.1 200 OK
```

The Web server responds in HTTP/1.1 with status code "200 OK," which indicates to the browser that the object was successfully fetched. Other codes are "403 Forbidden," (the server does not have permission to send the object to the client, and "404," (the server cannot find the object that the client requested).

```
Date: Thu, 30 Mar 2006 05:23:29 GMT
Last-Modified: Wed, 29 Mar 2006 16:22:05 GMT
Server: Apache/2.2.0
```

These lines allow the client to cache content efficiently. It tells the client what time the server thinks it is, and when the content was last modified. The server also identifies its product (Apache) and version (2.2.0), although this can be changed by the server administrator.

```
Accept-Ranges: bytes
Content-Length: 40
Connection: close
Content-Type: text/html; charset=UTF-8
<HTML><BODY>Hello, world!</BODY></HTML>
```

If needed to restart the transfer, the server tells the client in what form it can request portions of a file (in this case it accepts bytes). The server then tells the client to close the connection after the data is finished. The actual HTTP data follows, beginning with the line "Content-Type."

SMTP

SMTP is the mechanism by which most e-mail is sent over the Internet. SMTP uses TCP to transmit data exclusively, and in most situations the server uses port 25. The entire e-mail (headers and contents) is sent in one SMTP session. It is easy to emulate a SMTP session using the Telnet program to port 25 of an e-mail server.

Think of an SMTP connection the same as sending a memo through the regular mail. On the outside of the envelope is a return address and a destination. The return address and destination might also be repeated inside the envelope, but the mail carrier doesn't care about what is inside the envelope. In an SMTP connection, the message envelope is transmitted first, followed by the letter contents inside.

The following is an example of an SMTP conversation. The client sends in normal text, and the server responds in italics:

```
 220 example.org ESMTP Mail Service
HELO client.example.com
250 Ok
```

Upon connection, the server indicates its presence with a *banner* that includes the version of the e-mail server program, but can also be configured by the user to be an arbitrary banner, as long as it begins with the hostname of the server. The client says "HELO" to the server, and tells it what name it wants to go by. The **HELO** command is also used for clients that support advanced SMTP features such as encryption. The server acknowledges the client with an "Ok" response.

```
MAIL FROM:<person@example.com>

250 Ok
```

The client sends a **MAIL FROM** command, indicating the return address, which may or may not match the letter's contents. Your Mail User Agent or e-mail reader normally only show the address contained in the letter, disregarding the envelope.

```
RCPT TO:<anotherperson@example.org>
250 Ok
```

The client sends the destination of the envelope and the server acknowledges it.

At this point, you may see a "Relaying Denied" message, indicating that the server will not accept the e-mail. At the beginning of e-mail, systems were free to send e-mail directly to each other, or by relaying it through any other system on the Internet. However, this arrangement broke down in the 1990s when spam became a major issue on the Internet. Most systems now accept e-mail for themselves only, and relaying is generally only relevant to ISPs.

```
DATA
```

```
354 End data with <CR><LF>.<CR><LF>
Date: Wed, 29 Mar 2006 21:47:56 -0500
From: "person" <person@example.com>
To: anotherperson@example.com
Subject: Example E-Mail subject.

Example e-mail contents.

.

250 Ok: queued as C8243B4039
QUIT
221 Bye
```

The client sends the **DATA** command, telling the server that the contents of the letter will be transmitted. The e-mail's headers are normally repeated at this point, and it is from here that e-mail is communicated. To end the e-mail, the server instructs the client to send a linefeed, followed by a dot and another linefeed. The client politely issues the *QUIT* command, and the server bids the client farewell.

Protecting Against Sniffers

So far, you have learned what sniffing is and how it works. You have also learned some of the tricks that can be used by intruders for sniffing, and some not-so-foolproof methods of detecting sniffers. None of this sheds a positive light on your plight to protect your network and data. However, there are some methods on your network that offer protection against sniffing.

We talked earlier about using switches instead of hubs, and we learned the methods used to defeat switches. Using switches is a network best practice that allows increased performance and security. While switches present a barrier to casual sniffing, the best method of protecting your data is to use encryption, which is the best form of protection against traffic interception on public networks and internal networks. Intruders can still sniff the traffic, but the data appears unreadable. Only the intended recipient should be able to decrypt and read the data; however, some methods of encryption leave the packet headers in cleartext, thereby allowing intruders to see the source and destination addresses and map the network. However, the data contained within the packet is protected. Other forms of encryption also mask the header portion of the packet.

A VPN uses encryption and authentication to provide secure communication over an otherwise insecure network. VPNs protect the transmission of data over the Internet and over your internal network. However, if an intruder compromises either of the end nodes of a VPN, the protection is rendered useless. Different types of VPN families are not inter-

changeable, but they can be combined and used in multiples. The following list describes some of the VPN methods used today that protect your data against sniffing:

- **SSH** SSH is an application-level VPN that runs over TCP to secure client-to-server transactions. This is often used for system logins and to administer servers remotely, and is typically used to replace Telnet, FTP, and the BSD **r** commands. However, any arbitrary TCP protocol can be tunneled through an SSH connection and used for numerous other applications. SSH provides authentication using Rivest, Shamir, & Adleman (RSA) or Digital Signature Algorithm (DSA) asymmetric key pairs, and many encryptions options for protecting data and passwords sent over the network. The headers in an SSH session are not encrypted, so an intruder can still view the source and destination addresses.
- **Secure Sockets Layer (SSL)/Transport Layer Security (TLS)** SSL was originally developed by Netscape Communications to provide security and privacy to Internet sessions. It has been replaced by TLS, as stated in RFC 2246. TLS provides security at the transport layer and overcomes some security issues of SSL. It is used to encapsulate the network traffic of higher-level applications such as HTTP, Lightweight Directory Access Protocol (LDAP), FTP, SMTP, POP3, and IMAP. It provides authentication and integrity via digital certificates and digital signatures and the source and destination IP headers in a SSL session are not encrypted.
- **IP Security (IPSec)** IPSec is a network-level protocol that incorporates security into the IPv4 and IPv6 protocols directly at the packet level, by extending the IP packet header. This allows the ability to encrypt any higher-layer protocol. It has been incorporated into routing devices, firewalls, and clients for securing trusted networks to one another. IPSec provides several means for authentication and encryption, supporting a lot of public key authentication ciphers and symmetric key encryption ciphers. It can operate in *tunnel* mode to provide a new IP header that masks the original source and destination addresses in addition to the data being transmitted. Since IPSec uses protocols other than TCP and UDP, getting the IPSec traffic through a firewall or NAT device can be challenging.
- **OpenVPN** OpenVPN is a tunneling SSL VPN protocol, which can encrypt both the contents of a packet and its IP headers. OpenVPN uses a single TCP or UDP port; therefore, it can be easier to use in environments with challenging NAT and firewall architectures. Additionally, it can act as a virtual network bridge (a layer 2 VPN).

One-time passwords (OTP) are another method to protect against sniffing. S/key, One-time Passwords In Everything (OPIE), and other one-time password techniques protect against the collection and reuse of passwords. They operate using a challenge-response method, and a different password is transmitted each time authentication is needed. The pass-

words that a sniffer collects are eventually useless since they are only used once. Smart cards are a popular method of implementing one-time passwords. However, OTP technologies cannot help protect your password after you enter it.

E-mail protection is a hot topic for companies and individuals. Two methods of protecting e-mail (i.e., encrypting it in-transit and in storage) are Pretty Good Privacy (PGP) and Secure Multipurpose Internet Mail Extensions (S/MIME). Each of these methods also provides authentication and integrity using digital certificates and digital signatures.

Network Analysis and Policy

Before cracking open your newly installed network analyzer at work, read your company policy! A properly written and comprehensive "Appropriate Use" network policy will prohibit you from running network analyzers. Usually the only exception to this is if network analysis is in your job description. Also, just because you provide security consulting services for company clients does not mean that you can use your sniffer on the company network. However, if you are an administrator and allowed to legitimately run a sniffer, you can use it to enforce your company's security policy. If the policy on using sniffers is not clear in your organization, take the time to get permission in writing from the appropriate departments before using them or any other security-related tools. On the other hand, if your security policy prohibits using file-sharing applications (e.g., KaZaA, Morpheus, BitTorrent or messaging services such as Internet Relay Chat [IRC] or Instant Messenger [IM]), you could use a sniffer to detect this type of activity.

Also, if you provide security services for clients, be sure that using a sniffer is included in your Rules of Engagement. Be very specific about how, where, and when it will be used. Also, provide clauses (e.g., Non-Disclosure Agreements) that will exempt you from the liability of learning confidential information.

Ensure that your sniffing activities do not violate any laws against wiretapping. In many countries, wiretapping laws were enacted at a time when modems were the most complicated network access device, so the clarification of laws and related regulatory requirements can be complex and differ based on the situation and the parties involved.



CAUTION

Many ISPs prohibit using sniffers in their "Appropriate Use" policy. If they discover that you are using one while attached to their network, they may disconnect your service. The best place to experiment with a sniffer is on your home network that is not connected to the Internet. All you need is two computers with a crossover cable between them, or a virtual machine application. You can use one as a client, and install server services on the other (e.g., Telnet, FTP, Web, and e-mail).

NOTE

You can download packet traces from numerous Web sites and read them with your network analyzer to get used to analyzing and interpreting packets.

The HoneyNet Project at www.project.honeynet.org has monthly challenges and other data for analysis.

The Wireshark "wiki" also has many well-described capture files that are located at www.wiki.wireshark.org/SampleCaptures.

Summary

Network analysis is the key to maintaining an optimized network and detecting security issues. Proactive management can help find issues before they turn into serious problems and cause network downtime or compromise confidential data. In addition to identifying attacks and suspicious activity, your network analyzer can be used to identify security vulnerabilities and weaknesses and enforce your company's security policy. Sniffer logs can be correlated with IDSes, firewalls, and router logs to provide evidence for forensics and incident handling. A network analyzer allows you to capture data from the network (packet-by-packet), decode the information, and view it in an easy-to-understand format. Network analyzers are easy to find, often free, and easy to use; they are a key part of any administrator's toolbox.

This chapter covered the basics of networking, Ethernet, the OSI model, and the hardware that is used in a network architecture; however, it only scratched the surface. A good networking and protocol reference should be on every administrator's bookshelf. It will come in handy when you discover some unknown or unusual traffic on your network.

As an administrator, you should know how to detect the use of sniffers by intruders. You should keep up-to-date on the methods that intruders use to get around security measures that are meant to protect against sniffing. As always, you also need to make sure that your computer systems are up-to-date with patches and security fixes to protect against rootkits and other backdoors.

This chapter also covered a variety of methods used to protect data from eavesdropping by sniffers. It is important to remain up-to-date on the latest security technologies, encryption algorithms, and authentication processes. Intruders are constantly finding ways to defeat current security practices, thus more powerful methods are developed (e.g., cracking the Data Encryption Standard [DES] encryption scheme and its subsequent replacement with Triple Data Encryption Standard (3DES), followed by the Advanced Encryption Standard (AES).

Finally, remember the rule of network analysis—only do it if you have permission and the law is on your side. A curious, up-and-coming administrator could easily be mistaken for an intruder. Make sure you have permission, or use your own private network to experiment.

Solutions Fast Track

What is Network Analysis and Sniffing?

- ☑ Network analysis is capturing and decoding network data.
- ☑ Network analyzers can be hardware or software, and are available both free and commercially.
- ☑ Network analyzer interfaces usually have three panes: Summary, Detail, and Data.

- ☑ The five parts of a network analyzer are: hardware, capture driver, buffer, real-time analysis, and decode.

Who Uses Network Analysis?

- ☑ Administrators use network analysis for troubleshooting network problems, analyzing the performance of a network, and intrusion detection.
- ☑ When intruders use sniffers, it is considered a passive attack.
- ☑ Intruders use sniffers to capture user names and passwords, collect confidential data, and map the network.
- ☑ Sniffers are a common component of a rootkit.
- ☑ Intruders use sniffers to control backdoor programs.

How Does it Work?

- ☑ Ethernet is a shared medium that uses MAC or hardware addresses.
- ☑ The OSI model has seven layers and represents a standard for network communication.
- ☑ Hubs send out information to all hosts on the segment, creating a shared collision domain.
- ☑ Switches have one collision domain per port and keep an address table of the MAC addresses that are associated with each port.
- ☑ Port mirroring is a feature that allows you to sniff on switches.
- ☑ Switches make sniffing more difficult; however, the security measures in switch architectures can be overcome by a number of methods, thus allowing the sniffing of traffic designated for other computers.
- ☑ Sniffing wired traffic can be done with many kinds of NICs; wireless sniffing requires greater attention to hardware details such as chipset and drivers.

Detecting Sniffers

- ☑ Sometimes sniffers can be detected on local systems by looking for the PROMISC flag.
- ☑ There are several tools available that attempt to detect promiscuous mode using various methods.

- ☑ Carefully monitoring hosts, hub and switch ports, and DNS reverse lookups assists in detecting sniffers.
- ☑ Honeypots are a good method to detect intruders on your network who are attempting to use compromised passwords.
- ☑ New sniffers are smart enough to hide from traditional detection techniques.

Protocol Dissection

- ☑ DNS packets can use either TCP or UDP, depending on the purpose of the query and the amount of data transmitted.
- ☑ NTP data transmissions generally use UDP port 123 for both the client and server side ports.
- ☑ Multiple virtual HTTP servers can listen on one port. The Host: header indicates to the server which virtual server the client intended to connect with.
- ☑ ??SMTP connection can be emulated with a simple network program such as Telnet. If your SMTP server is left open on the Internet, it will eventually be used to send spam.

Protecting Against Sniffers

- ☑ Switches offer little protection against sniffers.
- ☑ Encryption is the best method of protecting your data from sniffers.
- ☑ SSH, SSL/TLS, and IPSec are all forms of VPNs that operate at various layers of the OSI model.
- ☑ IPSec tunnel mode can protect the source and destination addresses in the IP header by appending a new header.

Network Analysis and Policy

- ☑ Make sure you have permission to use a sniffer on a network that is not your own.
- ☑ Read the appropriate use policies of your ISP before using a sniffer.
- ☑ If you are hired to assess a computer network and plan to use a sniffer, make sure you have a non-disclosure agreement in place, because you may have access to confidential data.
- ☑ One-time passwords render compromised passwords useless.

- ☑ E-mail should be protected while in transit, and stored with some type of data encryption method.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: What can I do to protect my network from sniffers?

A: Proper network security comes by design, not just through action. Some argue that there is nothing you can do to make your network completely secure. A combination of network access controls like 802.1x, ubiquitous and opportunistic encryption, and strong policies and procedures will go a long way to protecting your network from sniffers and other security issues. Using several layers of security is known as defense-in-depth, and is a standard best practice for secure network architectures.

Q: What is this opportunistic encryption that you speak of, and where can I buy it?

A: Opportunistic Encryption is the practice where communication between two parties becomes encrypted, even when those parties cannot be assured of each other's identity. More information on opportunistic encryption is available at www.en.wikipedia.org/wiki/Opportunistic_encryption free of charge. Ubiquitous opportunistic encryption is the theory that any network communication that can be encrypted, should be.

Q: How can I ensure that I am sniffing network traffic legally?

A: The best way to ensure that your sniffing activities are legal is to solicit expert legal counsel. In general, you should be safe if all parties to the communication that you are sniffing acknowledge that they have no expectation of privacy on your network. These acknowledgements can be in employment contracts, and should also be set as banners so that the expectation of no privacy is reinforced. It is advised that you get authorization (in writing) from your employer to use sniffing software.

Q: Is a sniffer running a security breach on my network?

A: Possibly. Check the source of the sniffing activity and verify that the interception has been authorized. Hackers and other network miscreants use sniffers to assist themselves

in their work . It is best to design networks and other applications that are resilient to network sniffing and other security issues.

Q: Can I use a sniffer as an IDS?

A: While a sniffer can act similarly to an IDS, it is not designed as such. IDSes have threshold, alerting, and reporting systems that are beyond the design specifications for most sniffers.

Q: How do I use a sniffer to see traffic inside a VPN?

A: VPN traffic is normally encrypted and most sniffing software does not have the ability to read encrypted packets, even if you have the decryption key. The best place to see VPN traffic is outside of the VPN tunnel itself.

Chapter 8

Basics of Cryptography and Encryption

Solutions in this chapter:

- **Algorithms**
- **Concepts of Using Cryptography**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

You have seen in previous chapters how the open source community has created powerful sniffing tools. You have seen how they can be used either to administer your network or to attack it. Because these sniffing tools are open source, and because it is relatively easy to place a Linux host on your company network, you need to consider ways to minimize improper use of packet capturing tools. Encryption solutions, such as Secure Shell (SSH) and Kerberos, are common solutions to this problem.

Algorithms are the underlying foundation of cryptography; therefore, this chapter looks at the basics of algorithms, covering symmetric and asymmetric encryption and hashing concepts. This chapter then discusses the concepts of cryptography.

For as long as people have been writing down information, there has been the need to keep some information secret, either by hiding its existence or changing its meaning. The study of these methods is the science of *cryptography*. *Encryption*, a type of cryptography, refers to the process of scrambling information so that the casual observer cannot read it. What are algorithms and keys? An *algorithm* is a set of instructions for mixing and rearranging an original message, called *plaintext*, with a message key to create a scrambled message, referred to as *ciphertext*. Similarly, a cryptographic key is a piece of data used to encrypt plaintext to ciphertext, and ciphertext to plaintext, or both (depending on the type of encryption).

What does the word *crypto* mean? It has its origins in the Greek word *kruptos*, which means *hidden*. Thus, the objective of cryptography is to hide information so that only the intended recipient(s) can read it. In crypto terms, the hiding of information is called *encryption*, and when information becomes readable, it is called *decryption*. A cipher is used to accomplish the encryption and decryption. Merriam-Webster's Collegiate Dictionary defines *cipher* as "a method of transforming a text in order to conceal its meaning." The information that is being hidden is called *plaintext*; once it has been encrypted, it is called *ciphertext*. The ciphertext is transported, secure from prying eyes, to the intended recipient(s), where it is decrypted back into plaintext.

Finally, there are two different subclasses of algorithms: *block ciphers* and *stream ciphers*. Block ciphers work on "blocks" or chunks of text in a series. Just as a paragraph is composed of many sentences, plaintext is composed of many blocks, which are typically variable lengths of bits. In contrast, a stream cipher operates on each individual unit (either letters or bits) of a message.

Algorithms

Why are there so many algorithms? Why doesn't the world standardize on one algorithm? Given the large number of algorithms found in the field today, these are valid questions with no simple answers. At the most basic level, it's a classic case of tradeoffs between security, speed, and ease of implementation. Here, *security* indicates the likelihood of an algorithm to

stand up to current and future attacks, *speed* refers to the processing power and time required to encrypt and decrypt a message, and *ease of implementation* refers to an algorithm's predisposition (if any) to hardware or software usage. Each algorithm has different strengths and drawbacks, and none of them are ideal in every way. This section discusses the key algorithms, which fall into three main categories:

- Symmetric cryptography
- Asymmetric cryptography
- Hashing algorithms

What Is Encryption?

Encryption is a form of cryptography that "scrambles" plaintext into unintelligible ciphertext. Encryption is the foundation of such security measures as digital signatures, digital certificates, and the Public Key Infrastructure (PKI) that uses these technologies to make computer transactions more secure. Computer-based encryption techniques use keys to encrypt and decrypt data. A *key* is a variable (sometimes represented as a password) that is a large binary number—the larger, the better. Key length is measured in bits, and the more bits in a key, the more difficult the key will be to "crack." For example, a 40-bit key is considered insecure by today's standards, but it can have a value between 1 and 2^140 (1,099,511,627,776, over a trillion).

The key is only one component in the encryption process. It must be used in conjunction with an encryption *algorithm* (a process or calculation) to produce the ciphertext. Encryption methods are usually categorized as either symmetric or asymmetric, depending on the number of keys that are used. These two basic types of encryption technology are discussed in the following sections.

Symmetric Encryption Algorithms

The most widely used type of encryption is *symmetric encryption*, which is aptly named because it uses one key for both the encryption and decryption processes. Symmetric encryption is also commonly referred to as *secret-key encryption* and *shared-secret encryption*, but all terms refer to the same class of algorithms.

The reason why symmetric encryption systems are abundant is speed and simplicity. The strength of symmetric algorithms lies primarily in the size of the keys used in the algorithm, as well as the number of cycles each algorithm employs. The cardinal rule is "fewer is faster."

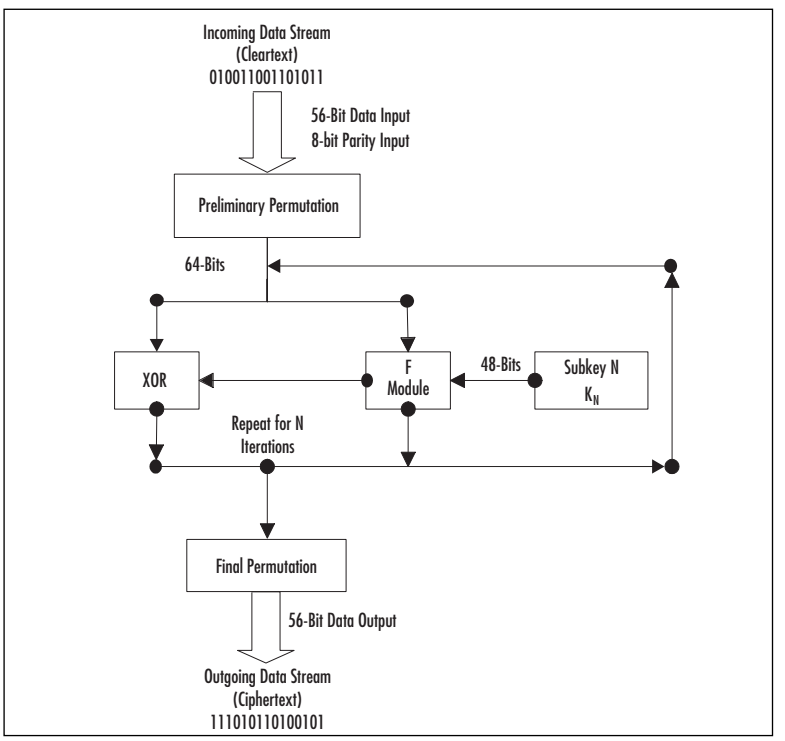
By definition, all symmetric algorithms are theoretically vulnerable to *brute-force*, which are exhaustive searches of all possible keys. Brute-force attacks involve methodically guessing what the key to a message may be. Given that all symmetric algorithms have a fixed key length, there are a large number of possible keys that can unlock a message. Brute-force

attacks methodically attempt to check each key until the key that decrypts the message is found. However, brute-force attacks are often impractical, because the amount of time necessary to search the keys is greater than the useful life expectancy of the hidden information. No algorithm is truly unbreakable, but a strong algorithm takes so long to crack that it is impractical to try. Because brute-force attacks originate from computers, and because computers are continually improving in efficiency, an algorithm that may be resistant to a brute-force attack performed by a computer today, will not necessarily be resistant to attacks by computers 5 to 10 years in the future.

Data Encryption Standard and Triple Data Encryption Standard

Among the oldest and most famous encryption algorithms is the Data Encryption Standard (DES), the use of which has declined with the advent of algorithms that provide improved security. DES was based on the *Lucifer algorithm* invented by Horst Feistel, which never saw widespread use. Essentially, DES uses a single 64-bit key—56 bits of data and 8 bits of parity—and operates on data in 64-bit chunks. This key is broken into 16 48-bit subkeys, one for each round, which are called *Feistel cycles*. Figure 8.1 gives a schematic of how the DES encryption algorithm operates.

Figure 8.1 Diagram of the DES Encryption Algorithm



Each round consists of a *substitution phase*, wherein the data is substituted with pieces of the key, and a *permutation phase*, wherein the substituted data is scrambled (re-ordered). *Substitution operations*, sometimes referred to as *confusion operations*, occur within *S-boxes*. Similarly, *permutation operations*, sometimes called *diffusion operations*, are said to occur in *P-boxes*. Both of these operations occur in the "F Module" of the diagram. The security of DES lies in the fact that since the substitution operations are non-linear, the resulting ciphertext does not resemble the original message. The permutation operations add another layer of security by scrambling the already partially encrypted message.

Triple DES (3DES) and DESX are methods that attempt to use the DES cipher in a way that increases its security. Triple DES uses three separate 56-bit DES keys as a single 168-bit key, though sometimes keys 1 and 3 are identical, yielding 112-bit security. DESX adds an additional 64 bits of key data. Both 3DES and DESX are intended to strengthen DES against brute-force attacks. It would take many years to decrypt 3DES encrypted date (depending on available computing power). However, 3DES is inefficient because it requires two to three times the processing overhead as a single DES.

Advanced Encryption Standard (Rijndael)

Because of its small key size of 56 bits, DES can't withstand coordinated brute-force attacks using modern cryptanalysis; dedicated machines can break DES within a day. Consequently, The National Institute of Standards and Technology (NIST) selected the Advanced Encryption Standard (AES) as the authorized Federal Information Processing Standard (FIPS) 197 for all non-secret communications by the U.S. government, which became effective in May 2002. AES has the following important characteristics:

- Private key symmetric block cipher (similar to DES)
- Stronger and faster than 3DES
- Life expectancy of at least 20 to 30 years
- Supports key sizes of 128 bits, 192 bits, and 256 bits
- Freely available to all; royalty free, non-proprietary, and not patented
- Small footprint. AES can be used effectively in memory and in central processing unit (CPU) limited environments such as Smart Cards

As a background note, the Rijndael algorithm (pronounced "rain doll") was selected by NIST from a group that included four other finalists: MARS, RC6, Serpent, and Twofish. It was developed by Belgian cryptographers Dr. Joan Daemen and Dr. Vincent Rijmen. NIST seems resistant to *side-channel attacks* such as *power-* and *timing-based attacks*, which are attacks against a hardware implementation, not against a particular algorithm. For example, power- and timing-based attacks measure the time it takes to encrypt a message or the minute changes in power consumption during the encryption and decryption processes.

Occasionally, these attacks are sufficient enough to allow hackers to recover keys used by the device.

So how does AES/Rijndael work? Instead of using Feistel cycles in each round like DES, Rijndael uses iterative rounds like International Data Encryption Algorithm (IDEA). Data operates on 128-bit chunks, which are grouped into four groups of 4 bytes each. The number of rounds is also dependent on the key size, such that 128-bit keys have 9 rounds, 192-bit keys have 11 rounds, and 256-bit keys have 13 rounds. Each round consists of a substitution step of one S-box per data bit, followed by a pseudo-permutation step in which bits are shuffled between groups. Then each group is multiplied out in a matrix fashion and the results are added to the subkey for that round.

IDEA

The European counterpart to the DES algorithm is the IDEA. Unlike DES, it is considerably faster and more secure. IDEA's enhanced speed is due to the fact that each round consists of simpler operations than in the Fiestel cycle in DES. IDEA uses simple operations like exclusive or (XOR), addition, and multiplication, which are more efficient to implement in software than the substitution and permutation operations of DES. Addition and multiplication are the two simplest binary calculations for a computer to perform. XOR is also a simple operation that returns a "1" when two inputs are different, and a "0" when both inputs are the same.

IDEA operates on 64-bit blocks with a 128-bit key, and the encryption/decryption process uses eight rounds with six 16-bit subkeys per round. The IDEA algorithm is patented both in the U.S. and in Europe, but free non-commercial use is also permitted. IDEA is widely recognized as one of the components of Pretty Good Privacy (PGP) (covered in Chapter 3). IDEA was developed in the early 1990s by cryptographer's James Massey and Xuejia Lai as part of a combined research project between Ascom and the Swiss Federal Institute of Technology.

Tools & Traps…

Assessing Algorithmic Strength

Algorithmic security can only be proven by its resistance to attack. Since many more attacks are attempted on algorithms that are open to the public, the longer an algorithm has been open to the public, the more attempts to circumvent or break it have occurred. Weak algorithms are broken rather quickly, usually in a matter of days or months, whereas stronger algorithms may be used for decades. However, the openness of the algorithm is an important factor. It's much more difficult to break an

Continued

algorithm (whether weak or strong) when its complexities are completely unknown. Thus, when you use an open algorithm, you can rest assured in its strength. This is opposed to a proprietary algorithm, which, if weak, may eventually be broken even if the algorithm itself is not completely understood by the cryptographer. Obviously, one should limit the trust placed in proprietary algorithms to limit long-term liability. Such scrutiny is the reason the inner details of many of the patented algorithms in use today (such as RC6 from RSA Laboratories) are publicly available.

Asymmetric Encryption Algorithms

The biggest disadvantage to using symmetric encryption algorithms relates to *key management*. In order to ensure confidentiality of communication between two parties, each communicating pair needs to have a unique secret key. As the number of communicating pairs increases, there is a need to manage a number of keys related to the square of the communicators, which quickly becomes a complex problem.

Asymmetric algorithms were developed to overcome this limitation. Also know as *public-key cryptography*, these algorithms use two different keys to encrypt and decrypt information. If cleartext is encrypted with an entity's public key, it can only be decrypted with its private key, and if it is encrypted with the private key, it can only be decrypted by the public key. The basic principle is that the *public key* can be freely distributed, while the *private key* must be held in strict confidence. The owner of the private key can encrypt cleartext to create cyphertext that can only be decoded with its public key (assuring the identity of the source), or it can use the private key to decrypt cyphertext encoded with its public key (assuring the confidentiality of the data). Although these keys are generated together and are mathematically related, the private key cannot be derived from the public key.

NOTE

Literally thousands of different cryptographic algorithms have been developed over the years. Cryptographic algorithms can be classified as follows:

- **Encryption Algorithms** Used to encrypt data and provide confidentiality
- **Signature Algorithms** Used to digitally "sign" data to provide authentication
- **Hashing Algorithms** Used to provide data integrity

Algorithms (ciphers) are also categorized by the way they work at the technical level (stream ciphers and block ciphers). This categorization refers to whether the algorithm is applied to a stream of data, operating on individual bits, or to an entire block of data. *Stream ciphers* are faster, because they work on smaller units of data. The key is generated as a *keystream,*

which is combined with the plaintext to be encrypted. RC4 is the most commonly used stream cipher. Another is ISAAC.

Block ciphers take a block of plaintext and turn it into a block of ciphertext. (Usually the block is 64 or 128 bits in size.) Common block ciphers include DES, CAST, Blowfish, IDEA, RC5/RC6, and SAFER. Most AES candidates are block ciphers.

Instead of relying on the techniques of substitution and transposition that symmetric key cryptography uses, asymmetric algorithms rely on the use of large-integer mathematics problems. Many of these problems are simple to do in one direction but difficult to do in the opposite direction. For example, it is easy to multiply two numbers together, but it is more difficult to factor them back into the original numbers, especially if the integers used contain hundreds of digits. Thus, in general, the security of asymmetric algorithms is dependent not upon the feasibility of brute-force attacks, but the feasibility of performing difficult mathematical inverse operations and advances in mathematical theory that may propose new "shortcut" techniques.

Asymmetric cryptography is much slower than symmetric cryptography. There are several reasons for this. First, it relies on exponentiation of both a secret and public exponent, as well as generation of a modulus. Computationally, exponentiation is a processor-intensive operation. Second, the keys used by asymmetric algorithms are generally larger than those used by symmetric algorithms, because the most common asymmetric attack (factoring) is more efficient than the most common symmetric attack (brute-force).

Because of this, asymmetric algorithms are typically used only for encrypting small amounts of information. In this section, we examine the RSA, Diffie-Hellman, and El Gamal algorithms.

Diffie-Hellman

The biggest problem in symmetric cryptography is the security of the secret key. Obviously, you cannot transmit the key over the same medium as the ciphertext, since any unauthorized parties observing the communications could use the key to decode the messages. Prior to the development of asymmetric cryptography and the Diffie-Hellman key exchange, secret keys were exchanged using trusted private couriers and other out-of-band methods.

In the mid-1970s, Whitfield Diffie and Martin Hellman published the Diffie-Hellman algorithm for key exchange, which allowed a secret key to be transmitted securely over an insecure line. This was the first published use of public-key cryptography, and one of the cryptography field's greatest advances. With the Diffie-Hellman algorithm, the DES secret key (sent with a DES-encrypted payload message) could be encrypted via Diffie-Hellman by one party, and decrypted only by the intended recipient.

Because of the inherent slowness of asymmetric cryptography, the Diffie-Hellman algorithm was not intended for use as a general encryption scheme. Rather, its purpose was to transmit a private key for DES (or a similar symmetric algorithm) across an insecure medium. In most cases, Diffie-Hellman is not used for encrypting a complete message, because it is much slower than DES, depending on implementation.

In practice, this is how a key exchange using Diffie-Hellman works:

1. Two parties agree on two numbers; one is a large prime number, the other is a small integer number. This can be done in the open, as it does not affect security.
2. Each of the two parties separately generate another number, which is kept secret. This number is equivalent to a *private key*. A calculation is made involving the private key and the previous two public numbers. The result is sent to the other party. This result is effectively a *public key*.
3. The two parties exchange their public keys. They then perform a calculation involving their own private key and the other party's public key. The resulting number is the *session key*. Each party should arrive at the same number.
4. The session key can be used as a secret key for another cipher, such as DES. No third party monitoring the exchange can arrive at the same session key without knowing one of the private keys.

Diffie-Hellman's greatest strength is that anyone can know either or both of the sender's and recipient's public keys without compromising the security of the message. Both the public and private keys are actually very large integers. The Diffie-Hellman algorithm takes advantage of complex mathematical functions known as *discrete logarithms*, which are easy to perform forward, but extremely difficult to inverse. Secure Internet Protocol (IPSec) uses the Diffie-Hellman algorithm in conjunction with the Rivest, Shamir, & Adleman (RSA) authentication to exchange a session key used for encrypting all traffic that crosses the IPsec tunnel.

El Gamal

The El Gamal algorithm is essentially an updated and extended version of the original Diffie-Hellman algorithm based on discrete logarithms. The security of the algorithm is roughly on par with that of the RSA algorithm. El Gamal has a few drawbacks, mainly its larger output and random input requirement. Encrypted El Gamal ciphertext is much longer than the original plaintext input, so it should not be used in places where bandwidth is a limiting factor, such as over slow wide area network (WAN) links. The El Gamal algorithm also requires a suitable source of randomness to function properly. It is worth noting that the Digital Signature Algorithm (DSA) was based on the El Gamal algorithm. DSA is a complementary protocol to RSA that is widely used in the OpenSSH implementation of the Secure Shell (SSH) protocol.

RSA

Shortly after the appearance of the Diffie-Hellman algorithm, Ron Rivest, Adi Shamir, and Leonard Adleman proposed another public key encryption system. Their proposal is now known as the RSA algorithm, named for the last initials of the researchers.

The RSA algorithm shares many similarities with the Diffie-Hellman algorithm in that RSA is also based on multiplying and factoring large integers. However, RSA is significantly faster than Diffie-Hellman, leading to a split in the asymmetric cryptography field that refers to Diffie-Hellman and similar algorithms as Public Key Distribution Systems (PKDS), and RSA and similar algorithms as Public Key Encryption (PKE). PKDS systems are used as session-key exchange mechanisms, while PKE systems are considered fast enough to encrypt small messages. However, PKE systems like RSA are not considered fast enough to encrypt large amounts of data such as entire file systems or high-speed communications lines.

Damage & Defense…

Understanding Asymmetric Key Sizes

RSA, Diffie-Hellman, and other asymmetric algorithms use larger keys than their symmetric counterparts. Common key sizes include 1024 bits and 2048 bits. The keys are this large because factoring, while still a difficult operation, is much easier to perform than the exhaustive key search approach used with symmetric algorithms. The slowness of PKE systems is also due to the larger key sizes. Since most computers can only handle 32 bits of precision, different "tricks" are required to emulate the 1024-bit and 2048-bit integers. However, the additional processing time is justified, since, for security purposes, 2048-bit keys are considered secure "forever."

Hashing Algorithms

Hashing is a technique in which an algorithm (also called a *hash function)* is applied to a portion of data to create a unique digital "fingerprint" that is a fixed-size variable. If anyone changes the data by so much as one binary digit, the hash function will produce a different output (called the *hash value* or a *message digest*) and the recipient will know that the data has been changed. Hashing can ensure integrity and provide authentication. The hash function cannot be "reverse-engineered"; that is, you can't use the hash value to discover the original data that was hashed. Thus, hashing algorithms are referred to as *one-way hashes*. A good hash function will not return the same result from two different inputs (called a *collision*). In other words, the *collision domain* of the function should be large enough to make it extremely

unlikely to have a collision. All of the encryption algorithms studied so far, both symmetric and asymmetric, are reversible, (i.e., they can be converted from cleartext to ciphertext and back again, provided the appropriate keys are used). However, there is no reversible function for hashing algorithms, so original material cannot be recovered. For this reason, hashing algorithms are commonly referred to as *one-way hashing functions*. However, irreversible encryption techniques are useful for determining data integrity and authentication.

Tools & Traps…

Understanding One-way Functions

What does it mean for a function to be considered one-way? First, consider the calculation of remainders in long division. Specifically, let's say that 5 divided by 2 equals 2 with a remainder of 1. The remainder part is known as a *modulus*, or *mod* for short, and is easy to calculate in one direction. But suppose the problem was "The remainder is 1, find the division problem." How would you know the correct answer? This is what is meant by a non-reversible function.

There is also a slightly more complex set of problems know as "clock arithmetic." Suppose that instead of having an infinite linear number line (i.e., 1, 2, 3…100, 101…) you had a number line that connected back on itself like a clock (i.e., 11, 12, 1, 2…10). On a clock, 5 + 3 is 8, but so is 5 + 15 and 5 – 9. Given the answer, you cannot derive a unique problem. Thus, clock arithmetic is another example of a one-way function.

Sometimes it is not necessary or even desirable to encrypt a complete set of data. Suppose someone wants to transmit a large amount of data, such as a CD image. If the data on the CD is not sensitive, they may not care that it is openly transmitted, but when the transfer is complete, they want to make sure the image you have is identical to the original image. The easiest way to make this comparison is to calculate a hash value on both images and compare results. If there is a discrepancy of even a single bit, the hash value of each will be radically different. Provided they are using a suitable hashing function, no two inputs will result in an identical output, or *collision*. The hashes created, usually referred to as *digital fingerprints*, are usually of a small, easily readable fixed size. Sometimes these hashes are referred to as *secure checksums*, because they perform similar functions as normal checksums, but are inherently more resistant to tampering.

Encrypted passwords are often stored as hashes. When a password is set for a system, it is generally passed through a hashing function and only the encrypted hash is stored. When a person later attempts to authenticate, the password is hashed and that hash is compared to the stored hash. If these are the same, they are authenticated, otherwise access is rejected. In theory, if someone were to obtain a password list for a system, it would be useless, since by

definition it is impossible to recover the original information from its hashed value. However, attackers can use dictionary and brute-force attacks by methodically comparing the output hash of a known input string to the stolen hash. If they match, the password has been *cracked*. Thus, proper password length and selection is highly desirable.

There are several different types of hashing, including division-remainder, digit rearrangement, folding, and radix transformation. These classifications refer to the mathematical process used to obtain the hash value. Let's take a quick look at these hashing algorithms:

- **Message Digest 4/Message Digest 5 (MD4/MD5)** The message digest (MD) class of algorithms were developed by Ron Rivest for use with digital signatures. They both have a fixed 128-bit hash length, but the MD4 algorithm is flawed and the MD5 hash has been adopted as its replacement.
- **Secure Hash Algorithm (SHA)** This hashing algorithm was created by the U.S. government (NIST and the National Security Agency [NSA]) and operates similarly to the MD algorithms. The most common is SHA-1, which is typically used in IPSec installations, and has a fixed hash length of 160 bits.

Notes from the Underground…

Using MD5 for Data Integrity

A few years ago, MD5 sums were used to verify that a distribution of OpenSSH, the popular open source SSH software, had been infected with a Trojan horse. The software itself was not trojaned, only the distribution files. Because certain operating systems such as FreeBSD automatically check MD5 sums of downloaded source against known MD5 sums of what the package should be, the trojaned files were discovered and removed from the distribution source within six hours of the infection.

Concepts of Using Cryptography

Cryptography is a word derived from the Greek *kryptos* ("hidden"), and the use of cryptography pre-dates the computer age by thousands of years. In fact, the history of cryptography was documented over 4000 years ago, where it was first allegedly used in Egypt. Julius Caesar even used his own cryptography called *Caesar's Cipher*. Basically, Caesar's Cipher rotated the letters of the alphabet to the right by three (e.g., *S* moves to *V* and *E* moves to *H*). By today's standards, the Caesar Cipher is extremely simplistic, but it served Julius just fine in his day. Keeping secrets has long been a concern of human beings, and the purpose

of cryptography is to hide information or change it so that it is incomprehensible to people for whom it is not intended. Cryptographic techniques include:

- **Encryption** Involves applying a procedure called an *algorithm* to plaintext to turn it into something that will appear to be gibberish to anyone who doesn't have the *key* to decrypt it.
- **Steganography** A means of hiding the existence of the data, not just its contents. This is usually done by concealing it within other, innocuous data.



NOTE

The words *cryptography* and *encryption* are often used interchangeably, but cryptography is a much broader term than encryption; encryption is a form of cryptography. In other words, all encryption is cryptography, but not all cryptography is encryption.

This section looks at some of the concepts and motivating factors behind the use of cryptography.

Confidentiality

The first goal of cryptography is confidentiality. Through the use of cryptography, users are able to ensure that only an intended recipient can "unlock" (decrypt) an encrypted message. Most modern algorithms are secure enough that those without access to the message "key" cannot read the message. Thus, it is extremely important to keep the secret key (when using symmetric algorithms) or private key (when using asymmetric algorithms) completely secret. If a secret or private key is compromised, the message essentially loses all confidentiality.



WARNING

Do not confuse confidentiality with authentication. Whether or not a person is allowed access to something is part of the authentication and authorization processes. An analogy: You are throwing a party. Because your house got trashed the last time, you want to ensure that only people who are invited attend. That is confidentiality, because you decided up front who would be invited. When the people come, they have to present an invitation to the doorman. That is authentication, because each guest had to show proof that they are who they claim to be. In general, confidentiality is planned in advance while authentication happens as a user attempts to access a system.

Integrity

Guaranteeing message integrity is another important aspect of cryptography. With cryptography, most asymmetric algorithms have built-in ways to validate that all the outputs are equivalent to the inputs. Usually, this validation is referred to as a *message digest*, and, on occasion, can be vulnerable to *man-in-the-middle (MTM) attacks*. (

Damage & Defense…

Principles of Cryptography

Cryptosystems are considered either weak or strong with the main difference being the length of the keys used by the system. In January 2000, U.S. export controls were relaxed. Now, strong (not military grade) cryptography can be exported, as long as the end user or customer does not belong to a terrorist organization or an embargoed country (e.g., Cuba, Iran, Iraq, Libya, North Korea, Serbia, Sudan, and Syria). DES was originally designed so that the supercomputers owned by the NSA could be used for cracking purposes, working under the premise that no other supercomputers of their sort are in the public hands or control.

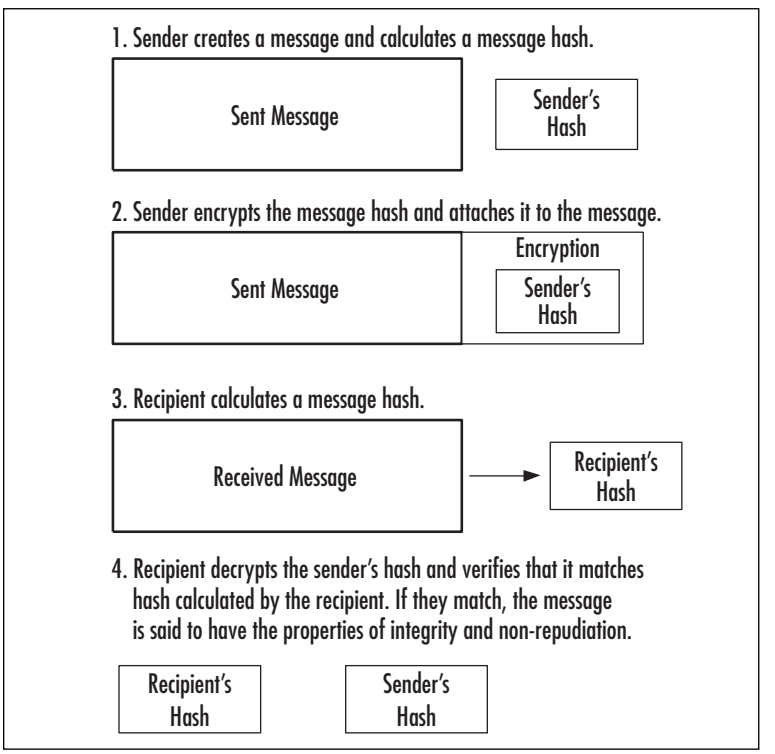
Strong cryptography always produces ciphertext that appears random to standard statistical tests. Because keys are generated for uniqueness using robust random number generators, the likelihood of their discovery approaches zero. Rather than trying to guess a key's value, it's far easier for would-be attackers to *steal* the key from where it's stored, so extra precautions must be taken to guard against such thefts.

Cryptosystems are similar to currency—people use them because they have faith in them. You can never *prove* that a cryptosystem is unbreakable (it's like trying to prove a negative), but you can demonstrate that the cryptosystem is *resistant* to attacks. In other words, there are no perfect cryptosystems in use today, but with each failed attempt at breaking one, the strength of the faith grows. The moment a cryptosystem is broken (and knowledge of that is shared), the system collapses and no one will use it anymore. The strongest systems resist all attacks on them and have been thoroughly tested for assurances of their integrity. The strength of a cryptosystem is described in the size and the secrecy of the keys that are used, rather than keeping the algorithm itself a secret. In fact, when a new cryptosystem is released, the algorithms are also released to allow people to examine and try to create an attack strategy to break it (called *cryptanalysis*). Any cryptosystem that hasn't been subjected to brutal attacks should be considered suspect. The recent announcement by the NIST of the new AES to replace the aging DES system (described earlier), underscores the lengths to which cryptographers will go to build confidence in their cryptosystems.

Digital Signatures

Digital signatures serve to enforce data integrity and non-repudiation. A digital signature ensures that the message received was the message sent, because a hash was performed on the original message using a hashing algorithm. The hash value created by this process is encrypted by the author's private key and appended to the message. To verify that the message has not been modified, the recipient uses the author's public key to decrypt the hash created by the author. The recipient also creates a hash of the message body. If the recipient's hash matches the hash created by the author of the message, the recipient knows that the message is unaltered. Refer to Figure 8.2 for the digital signature verification process.

Figure 8.2 Digital Signature Verification Process



MITM Attacks

Some types of asymmetric algorithms are immune to MITM attacks, which are only successful the first time two people try to communicate. When a third party intercepts the communications between the two trying to communicate, the attacker uses his own credentials to impersonate each of the original communicators.

Beware of the key exchange mechanism used by any PKE system. If the key exchange protocol does not authenticate at least one and preferably both sides of the connection, it

may be vulnerable to MITM-type attacks. Authentication systems generally use some form of digital certificates (usually X.509), and require a PKI infrastructure.

Also, note that MITM-based attacks can only occur during the initial correspondence between two parties. If their first key exchange goes unimpeded, then each party will authenticate the other's key against prior communications to verify the sender's identity.

Bad Key Exchanges

Because there isn't any authentication built into the Diffie-Hellman algorithm, implementations that use Diffie-Hellman-type key exchanges without some sort of authentication are vulnerable to MITM attacks. The most notable example of this type of behavior is the SSH-1 protocol. Since the protocol itself does not authenticate the client or the server, it's possible for someone to cleverly eavesdrop on the communications. This deficiency was one of the main reasons that the SSH-2 protocol was completely redeveloped from SSH-1. The SSH-2 protocol authenticates both the client and the server, and warns of or prevents any possible MITM attacks, depending on configuration, so long as the client and server have communicated at least once. However, even SSH-2 is vulnerable to MITM attacks prior to the first key exchange between the client and the server.

As an example of a MITM-type attack, consider that someone called Al is performing a standard Diffie-Hellman key exchange with Charlie for the very first time, while Beth is in a position such that all traffic between Al and Charlie passes through her network segment. Assuming Beth doesn't interfere with the key exchange, she will not be able to read any of the messages passed between Al and Charlie, because she will be unable to decrypt them. However, suppose that Beth intercepts the transmissions of Al and Charlie's public keys and she responds to them using her own public key. Al will think that Beth's public key is actually Charlie's public key and Charlie will think that Beth's public key is actually Al's public key.

When Al transmits a message to Charlie, he will encrypt it using Beth's public key. Beth will intercept the message and decrypt it using her private key. Once Beth has read the message, she encrypts it again using Charlie's public key and transmits the message on to Charlie. She may even modify the message contents if she so desires. Charlie then receives Beth's modified message, believing it to come from Al. He replies to Al and encrypts the message using Beth's public key. Beth again intercepts the message, decrypts it with her private key, and modifies it. Then she encrypts the new message with Al's public key and sends it on to Al, who receives it and believes it to be from Charlie.

Clearly, this type of communication is undesirable, because a third party not only has access to confidential information, but she can also modify it at will. In this type of attack, no encryption is broken because Beth does not know either Al or Charlie's private keys, so the Diffie-Hellman algorithm isn't really at fault. Beware of the key exchange mechanism used by any PKE system. If the key exchange protocol does not authenticate at least one and preferably both sides of the connection, it may be vulnerable to MITM-type attacks. Authentication systems generally use some form of digital certificates (usually X.509), such as those available from Thawte or VeriSign.

Authentication

Is the receiver able to verify the sender? The answer depends on the type of encryption. In cases of symmetric cryptography, the answer is no, but in cases of asymmetric cryptography, the answer is yes. With symmetric cryptography, anyone with access to the secret key can both encrypt and decrypt messages. Asymmetric cryptography can authenticate a sender by their private key, assuming that the key is kept private. Because each person is responsible for their own private key, only that person is able to decrypt messages encrypted with their public key. Similarly, only those persons can sign messages with their private key that are validated with their public key.

Non-Repudiation

Asymmetric cryptography ensures that an author cannot refute that they signed or encrypted a particular message once it has been sent, assuming the private key is secured. Again, this goes back to the fact that an individual should be the only person with access to their private key. If this is true, only that person could sign messages with their private key and therefore, by extension, all messages signed with their private key originated with that specific individual.

Access Control

Additionally, in limited ways, cryptography can provide users with some access control mechanisms. Some systems can provide access control based on key signatures. Similar systems use X.509 certificates in the same manner. The idea is that, based on a certificate presented by a user that has been signed by that user, a particular user can be identified and authenticated. Once the authentication has occurred, software access controls can be applied to the user.

One-time Pad

There is a type of cryptography that has been mathematically proven to be unbreakable. The concept is called the *one-time pad (OTP)*. It requires you to use a series of random numbers equal in length to the message you want to send. The problem with using this type of cryptography is that both sides need access to the random number generator, and the random number listings can never be reused. A suitable source of randomness that is truly random and unpredictable to put the concept to use has not been found. Considering that OTP's were created almost 100 years ago, far before most modern cryptography techniques, and have been used in the military and intelligence communities for many years, it is a very interesting concept.

The OTP algorithm is actually a Vernam cipher, which was developed by AT&T in 1917. The Vernam cipher belongs to a family of ciphers called stream ciphers, since they

encrypt data in continuous stream format instead of the chunk-by-chunk method of block ciphers. There are two problems with using the OTP, however: You must have a source of truly random data, and the source must be bit-for-bit as long as the message to be encoded. You also have to transmit both the message and the key (separately), the key must remain secret, and the key can never be reused to encode another message. If an eavesdropper intercepts two messages encoded with the same key, then it is trivial for the eavesdropper to recover the key and decrypt both messages. The reason OTP ciphers are not used more commonly is the difficulty in collecting truly random numbers for the key and the difficulty of the secure distribution of the key.

Summary

This chapter examined many of the common cryptography algorithms and concepts that help apply cryptography in situations where it is necessary and effective.

It discussed three different classes of algorithms, including symmetric (also known as secret key), asymmetric (also known as public key), and hashing algorithms.

Specifically, the symmetric cryptography algorithms studied included DES, 3DES, AES (Rijndael), and IDEA. The most important aspects of these symmetric algorithms is that they use a single key for both encryption and decryption, are generally fast, use small key sizes (generally around 128 bits), and are vulnerable to brute-force attacks.

The three asymmetric algorithms studied were RSA, Diffie-Hellman, and El Gamal. Asymmetric algorithms use a combination of keys for encryption and decryption, are relatively slow, use large key sizes (greater than 512 bits), and are vulnerable to factoring-based attacks and mathematical discoveries. Some of the hashing algorithms looked at included MD4, MD5, and SHA-1. Hashing algorithms are most often used to verify file integrity and to encrypt system passwords.

Also explored were some of the concepts behind cryptography, including confidentiality, integrity, authentication, and non-repudiation. Confidentiality is the idea that information should only be accessible by those with a "need to know," and authentication is the act of verifying that a person or process is whom they claim to be. Integrity means that a message has remained unmodified since the author sent it, and non-repudiation is a corollary of integrity that prevents an author from denying that a message or part of its contents were sent. Some of these concepts also tie into the discussions of digital signatures. Digital signatures are a public key cryptography application that uses the concepts of confidentiality, integrity, and non-repudiation to create an accountable messaging system. Some cryptography attacks were discussed, such as the MITM attack, which is a common attack against asymmetric encryption that allows a third party to eavesdrop on the initial communications between two parties.

Solutions Fast Track

Algorithms

- ☑ For Symmetric algorithms are relatively fast and use only a single key for both encryption and decryption. A single key for each communicating pair leads to complex key management issues. Some examples of symmetric algorithms are DES, 3DES, AES, and IDEA.

- ☑ Asymmetric algorithms use a separate key for both the encryption and decryption processes, are relatively slow, and the concepts are newer than those of symmetric algorithms. Some examples of asymmetric algorithms include Diffie-Hellman, RSA, and El Gamal.
- ☑ Hashing algorithms are used to create secure fixed-length checksums, which are often used for integrity verification. Some examples include MD4, MD5, and SHA-1.

Concepts of Using Cryptography

- ☑ Digital signatures are an application of public-key cryptography that can prove a message came from a specific person and verify that the text of the recipient's message matches the text of the sender's message.
- ☑ Confidentiality within the context of cryptography is the idea that information can only be accessed by people with a need to know.
- ☑ Integrity within the context of cryptography is the idea that a message has been received in its original unmodified form after transmission.
- ☑ Authentication is the act of verifying that a person or process is whom it claims to be.
- ☑ Non-repudiation is a subset of integrity that prevents an author from denying that he or she wrote a particular message.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Are the concepts of confidentiality, integrity, and authentication limited only to cryptography?

A: Absolutely not! The concepts of confidentiality and integrity are part of the CIA principles and you will find them turning up often in information security. In fact, a large portion of information security is concerned with keeping information on a need-to-know basis (confidentiality), making sure that you can trust information that you have (integrity). A non-cryptography-related example of each would be operating system access controls, file system verification tools like Tripwire, and firewall rules. That is not to say that authentication is not important, however. If you cannot determine whether or not people or processes are who they claim to be, your other security precautions become useless.

Q: All of the algorithms looked at in this chapter are theoretically vulnerable in some way, either by brute-force attacks or mathematical advances. Why are they used?

A: Although none of the algorithms in this chapter are 100 percent unbreakable, they are an effective method for protecting confidential information. The main principle is that with algorithms like 3DES, AES, and RSA, decrypting a single piece of information will take tens, hundreds, or thousands of years. By that time, it is assumed that the information will no longer be valuable.

Q: Are there any cryptography techniques which are 100 percent secure?

A: Yes. Only the OTP algorithm is absolutely unbreakable if implemented correctly. The OTP algorithm is actually a Vernam cipher, which was developed by AT&T in 1917. The Vernam cipher belongs to a family of ciphers called *stream ciphers*, since they encrypt data in continuous stream format instead of the chunk-by-chunk method of block ciphers. There are two problems with using the OTP, however: You must have a source of truly random data, and the source must be bit-for-bit as long as the message to be encoded. You also have to transmit both the message and the key (separately), the key must remain secret, and the key can *never* be reused to encode another message. If an eavesdropper intercepts two messages encoded with the same key, then it is trivial for the

eavesdropper to recover the key and decrypt both messages. The reason OTP ciphers are not used more commonly is the difficulty in collecting truly random numbers for the key (as mentioned in one of the sidebars for this chapter) and the difficulty of the secure distribution of the key.

Q: How long are DES and 3DES expected to remain in use?

A: Most systems are capable of either 3DES or AES encryption. Both are considered secure and reliable, and no computer system in use today can crack them for the foreseeable future. DES, on the other hand, can already be broken within a day, so its use is highly discouraged. With high performance machines and dedicated processors and card, there should be no reason to use DES.

Q: Why was the Content Scrambling System (CSS), the encryption technology used to protect DVDs from unauthorized copying, able to be broken so easily?

A: Basically, DVD copy protection was broken so easily because one entity, Xing Technologies, left their key lying around in the open, which as we saw in this chapter, is a cardinal sin. The data encoded on a DVD-Video disc is encrypted using the CSS algorithm, which can be unlocked using a 40-bit key. Using Xing's 40-bit key, hackers were able to brute force and guess at the keys for over 170 other licensees at a rapid pace. That way, since the genie was out of the bottle, so to speak, for so many vendors, the encryption for the entire format was basically broken. With so many keys to choose from, others in the underground had no difficulty in leveraging these keys to develop the CSS program, which allows data copied off of the DVD to be saved to another media in an unencrypted format. Ultimately, the CSS scheme was doomed to failure. You can't put a key inside millions of DVD players, distribute them, and not expect someone to eventually pull it out.

Chapter 9

Perimeter Security, DMZs, Remote Access, and VPNs

Solutions in this chapter:

- **Firewall Types**
- **Firewall Architectures**
- **Implementing Firewalls**
- **Providing Secure Remote Access**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

When it comes to securing networks, the first items that come to mind are firewalls, which are the primary gatekeepers between an organization's internal network and the outside world. While a properly implemented firewall can be one of the most effective security tools in your arsenal, it shouldn't be the only tool. The adage "defense-in-depth" means that you should have multiple layers of security. Using a defense-in-depth configuration, if one component of your defense failed or was defeated, there would still be a variety of other fallbacks to protect your network. With the availability of increasingly affordable firewalls such as the popular Linksys cable/digital subscriber line (DSL) router, using the free firewall alternatives may not be as attractive for some. With a little effort, however, you will find the free alternatives are more configurable, allowing greater flexibility and control than the "home office" grade offerings.

This chapter focuses on securing your network perimeter. Remember that although the most common way to implement a firewall is between an internal network and the outside world (often the Internet), you should not limit yourself to placing firewalls only on the network edge. A firewall should be in any place you want to restrict the flow of traffic. With the current trend of security breeches originating from the inside of the network (often employees or ex-employees), companies are increasingly relying on firewalls to isolate and filter traffic between portions of the internal network.

This chapter reviews some basic firewall concepts and briefly discusses the different architectural ways to implement a firewall. The meat of this chapter discusses the installation and configuration of free firewalls to run on your Linux-based systems. Finally, once the network edge has been adequately secured, we discuss how to create controlled, secure paths through the perimeter for remote connectivity, including administrative access or remote office/work from home scenarios.

Firewall Types

No discussion of firewalls would be complete without a discussion of the different types of firewalls. This is particularly true in this context, because it allows you to better understand exactly where in the spectrum the free firewall offerings lie. In the networking sense, a firewall is basically any component (software or hardware) that restricts the flow of network traffic. This is a sufficiently broad definition to allow for all of the various ways people have chosen to implement firewalls. Some firewalls are notoriously limited in capability and others are extremely easy to use.

Within the realm of firewalls there are many different ways to restrict network traffic. Most of these methods vary in the level of intelligence that is applied to the decision-making process. For example, to permit or deny traffic based on which network device is the sender or recipient, you would use a *packet-filtering firewall*. In reality, even the simplest

packet filtering firewalls can typically make decisions based on the source Internet Protocol (IP) address, the destination IP address, and the source and/or destination port number. While this type of firewall may sound overly simplistic, consider if you have a server running a Web site for use on the Internet. In all likelihood, the only traffic that you need to allow to the server uses a destination port of Transmission Control Protocol (TCP) 80 or 443; thus, you could configure your firewall to permit only that traffic. These ports are used for HTTP and HTTPS, respectively Because the server is available for the Internet, you can't filter traffic based on the source address or source port, which will be different for each connection.

The primary drawback with a simple packet filter is that the packet filtering firewall has to rely on very primitive means to determine when traffic should be allowed (e.g., synchronous [SYN] or acknowledgement [ACK] bits being set). While this was adequate in the early days of the Internet when security was not as big of a concern, it won't work any more. It is trivial to set the bits on the packet using freely available software to make the traffic look like it is a reply to another connection. Thus the *stateful inspection firewall* was born of necessity. This type of firewall monitors all connections (inbound or outbound), and as the connection is permitted (based on the firewall's configured rules) it enters this connection into a table. When the reply to this connection comes back, even if the reply uses a port that the firewall was not previously configured to permit, it can intelligently realize the traffic is a response to a permitted session and permit the traffic.

Unfortunately, as the firewalls get better so do the methods hackers use to circumvent them. Suppose you have configured your firewall perfectly and there are no holes*:* every permitted port is one you expressly want to allow. Using the previous example, no traffic is allowed to the Web server except Web traffic. Sounds good, but the problem is, if the firewall is completely secure, the server might not be. Flaws in the Web server software could allow the attacker to send the server an HTTP request that is 10,000 characters long, overflowing the buffers and allowing the attacker to execute the code of his choice. The packets used to transport the 10,000-character HTTP request are all legal TCP packets as far as the firewall is concerned*:* therefore, it would permit them to pass through to the Web server. The next step in firewall evolution serves to combat this type of attack. These types of firewalls are *application gateways*, or layer 7 firewalls.

This type of firewall not only filters network traffic based on the standard network parameters, but they also understand the higher layer protocol information contained within the packet, in this example HTTP. The firewall itself knows what a legitimate HTTP request looks like and can filter out a malformed or malicious request even though, from a network perspective, it might otherwise be a permitted packet. There is a down side to this type of approach, which is that the firewall must be programmed with all the same intelligence needed to filter normal traffic, plus the firewall must fully understand the protocols it is inspecting. This means additional programming for any protocol you want the firewall to understand. Most of the major commercial application gateways offer support for the major

protocols such as HTTP, File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP).

With all of this information circulating in your head, you're probably wondering which type is available for free? Generally speaking, you can find many free varieties of firewalls that perform some type of stateful inspection. Application layer gateways are not readily available for free. In reality, few organizations have the funds to use application gateways extensively. One ramification of *not* using an application gateway is that you need to ensure that the service that is exposed to un-trusted traffic is configured as securely as possible and that the server itself is hardened against attack. Keeping the service patches up-to-date will help reduce the odds that an application-level attack will be successful.

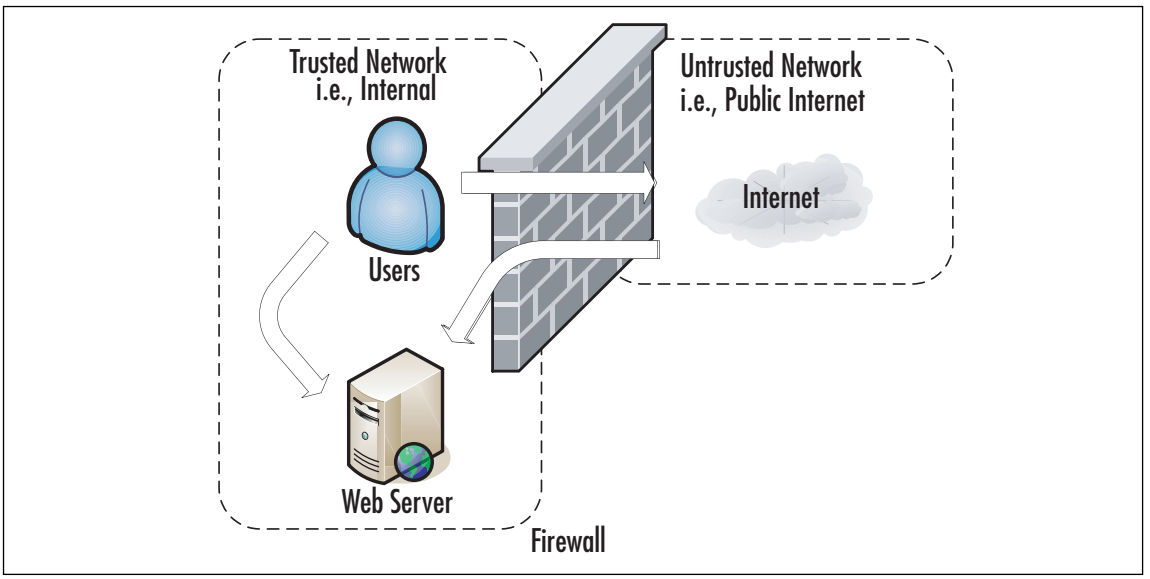
Firewall Architectures

The most securely configured firewall in existence will not provide much protection if a network was not designed properly. For example, if the firewall was installed into an environment that allows an alternate network path that bypasses the firewall, the firewall would only be providing a false sense of security. This is an architectural error that would render the firewall useless. In short, where the firewall is implemented is every bit as important as how it is implemented. The first step to installing anything is always planning. What follows is a discussion of the most common firewall architectures, in increasing order of security. Remember, these sections are discussing firewall architectures independent of the firewall type. For example, you could use a packet-filtering firewall, a stateful inspection firewall, or an application gateway in any of the designs discussed in the next section.

Screened Subnet

A *screened subnet* is the simplest and most common firewall implementation. Most small businesses and homes use this type of firewall (see Figure 9.1). This design places the firewall on the edge of your network, dividing everything (from the firewall's point of view) into internal and external, with nothing in between.

Figure 9.1 Screened Subnet Firewall



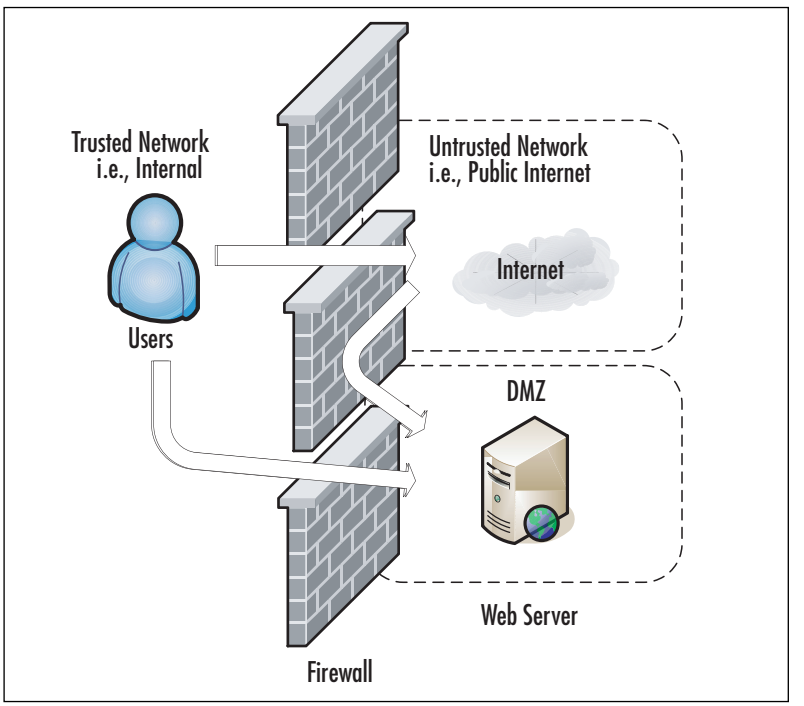
The screened subnet firewall (or *edge firewall*) is as straightforward as you can get. Internet users who need access to an internal server (e.g., Web, FTP, SMTP, and so on) must traverse the firewall to do so. Internal users needing access to those same servers would be able to access them directly. Internet traffic not destined for any Web-based server would be blocked at the firewall to prevent attacks on internal systems. All internal users must also traverse firewalls to access the Internet. This is the same type of firewall architecture you would have at home with a small network behind a Linksys router. This configuration has several advantages. The primary advantage is simplicity. With only two interfaces, the Access Control Lists (ACLs) (the filters that define the criteria for permitting or denying traffic) are much simpler.

Although this configuration is cost effective and simple to implement, it is not without it's drawbacks. In this arrangement, the hacker has several chances to penetrate your network. If he or she can find a security hole in the firewall, or if the firewall is improperly configured, he or she might be able to gain access to the internal network. Even if the firewall is executed flawlessly, the hacker has a second opportunity to gain access. If the hacker can compromise any available Web-based services and take control of the servers, he or she would then have an internal system from which to launch additional attacks. Finally, if the servers are critical to the business function, by allowing the internal users to access them without going through the firewall, you may loose some audit capability that the firewall might otherwise offer. By far the biggest security weakness in this configuration is that if you are exposing any Web-based services: the servers hosting those services will be attacked frequently, and a compromise of one of those servers may expose your entire network.

One-Legged

The one-legged demilitarized zone (DMZ) still has the advantage of cost, because you are building a DMZ using only a single firewall (see Figure 9.2). Commonly, the firewall interfaces are called Internal or Inside, External or Outside, and DMZ. (See Chapter 10 for details on configuring a Linux Bastion server.)

Figure 9.2 One-legged DMZ



With this type of configuration you get to keep the low cost benefit, but add some isolation to your Internet-based servers. Internal users must traverse the firewall to access the servers or the Internet. External users must traverse the firewall to access the Web-based services. The real strength of this type of configuration is that if the servers that are hosting the Web-based services are compromised, the hacker still needs to contend with the firewall to continue attacking the internal network. As an added feature, because all users (internal or external) must traverse the firewall to access the Web-based servers, you may gain a higher degree of auditing from the firewall logs. If you wanted to provide even further isolation, assuming you have the available interfaces on the firewall, you could implement a separate DMZ for each Web-based server you needed.

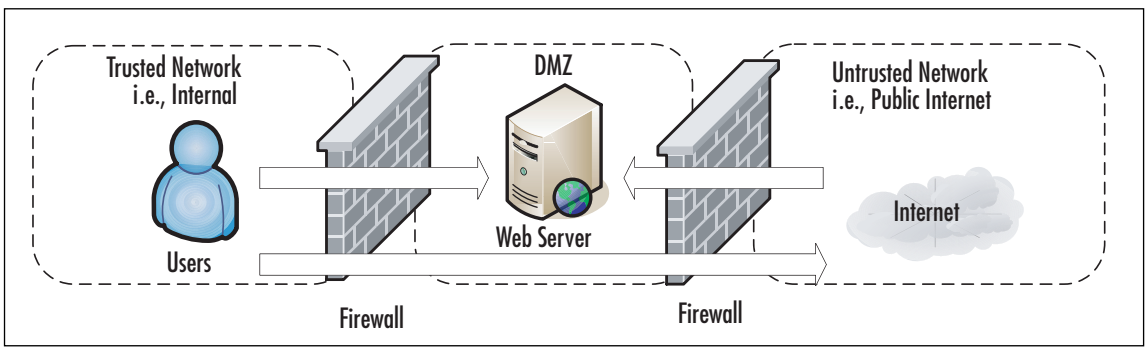
The only real disadvantages to this configuration are complexity, and to a small degree, cost. As you add interfaces to the firewall, the configuration will become more complex. Not

only does this complexity add to the time and labor for configuration and maintenance, it also increases the chance that an error could be made in the configuration. As you add interfaces there will often be additional costs associated with them. In most cases this cost will be minor and far less than an additional firewall, but with some high-speed interfaces, they can become very costly. Lastly, though many would consider it minor, with this configuration, if the firewall itself is defeated, the entire network is open to attack. Of course the solution to such paranoid thinking is costly.

True DMZ

The true DMZ is generally considered the most secure of firewall architectures. With this design, there is an external and internal firewall. Between the two is sandwiched any Internet accessible devices (see Figure 9.3).

Figure 9.3 True DMZ



Internet traffic is only permitted to a server in the DMZ, and only on the port that server is listening on. For example, if you had a Web server in the DMZ and an FTP server in the DMZ, traffic with a destination port of 80 would only be permitted to the Web server. For users accessing the same servers, the same rules would apply. Internal users would have to have permission through both firewalls to access the Internet. Obviously, this type of design costs more, typically double, but that cost buys you increased security. In a true DMZ, if the Web server is compromised the hacker is still trapped between two firewalls. For those who want to go the extra mile, the inside and outside firewalls can be of different types (say Cisco Private Internet Exchange [PIX] and Linux netfilter). In this way, a hacker that finds a security hole in one firewall is unlikely to be able to apply the same techniques to the other firewall.

With all of the basics out of the way, you will be in a better position to make informed decisions when it comes time to propose and implement a firewall solution for your network. Bear in mind, while this chapter covers the basics of firewalls, there are entire volumes

(such as *Designing and Building Enterprise DMZs* by Syngress Publishing, 2006) that explore the topic of firewall architectures, DMZ design, and implementation.

Tools & Traps…

Accidents Happen

I saw a corporate firewall/DMZ with a connection that allowed traffic to completely bypass their Internet firewall. I do not know why this happened, because the organization was not lacking properly trained networking professionals. These types of errors could occur because someone didn't analyze the implications of the changes adequately. Perhaps it was a "rush" to install some connectivity, or an emergency repair, or even a "temporary" fix. All of these things would indicate poor change control procedures. It is also possible that someone didn't realize the complete layout of the network when they made the connection in question, which could indicate inadequate network documentation among other things. In any case, these were trained professionals who should have known better, but accidents happen to the best of us.

Implementing Firewalls

When it comes to selecting a firewall there are a host of factors to consider. For commercial offerings there is the up front cost in addition to ongoing maintenance costs, which in comes cases can be considerable. Some commercial offerings run on their own base system (e.g., Cisco PIX). The underlying Linux system has been so heavily modified it is now considered proprietary. In the case of a Linux firewall, you also have the option of installing the firewall software on a Compact Disk - Read Only Memory (CD-ROM) or pen drive. These steps are discussed in more detail in the following sections, along with specific configuration examples for setting up a free firewall on both Linux and Windows.

Hardware versus Software Firewalls

Another consideration is whether the firewall decision-making logic is run as software that sits on top of another functional system, or if the firewall is a dedicated piece of hardware. In the case of a Cisco PIX firewall, the smallest models are the size of a small cigar box and there is no OS other than the PIX software. This is a dedicated hardware device used to perform the firewall function, also called a *firewall appliance*. The other alternative is that the firewall is not a dedicated box, but a software component. Many popular firewalls take this approach as well, such as a *checkpoint firewall* that can be installed on top of a Windows

system. Of these two approaches, if you want a free solution the choice is made for you. I know of no free hardware-based firewalls, so you will be using a software firewall.

Configuring netfilter

When it comes to Linux-based firewalls, there is only one choice, which is netfilter. This is partially because it was the best option available for the longest time. Since version 2.4, however, netfilter has been built into the Linux kernel. Even many commercial firewalls are running a modified Linux OS with netfilter inside their own custom case. netfilter is the underlying software that makes up the built-in firewall on Linux systems. It is the netfilter component that reads the contents of the network packet and permit or deny network traffic. Many times people incorrectly refer to the firewall as iptables, or prior to that, ipchains. In fact, iptables is the software command that is used to configure the rules that netfilter uses to make decisions to permit or deny traffic and ipchains is the previous version of iptables. Even after you have settled on using Linux as your base OS for your firewall, there are some additional choices to make before you start any configuring.

Choosing a Linux Version

While all versions of Linux share some common characteristics, there will be differences. Depending on the specific Linux distribution, the differences could be significant and each distribution will likely offer some different sets of software packages. An excellent source of information on the different distributions is www.distrowatch.com. This site includes a brief summary of what the distribution is trying to accomplish, and includes links to the home page and download locations. Because there are so many free versions of Linux available, it doesn't cost anything but the time to download and install several different versions and see which one you like. In the following examples I use a base system of Fedora core 5, which is the free version of the Red Hat Enterprise Linux that many companies use. I chose this distribution because, being one of the oldest and most well-established Linux distributions, there is extensive support documentation available if you need it. If you just want to see if Linux is something you want to work with, try a live CD such as SLAX. When it comes to choosing the specific version of Linux you want to use, this decision must be made in parallel with choosing an installation media, because not all versions are supported on all media.

Choosing Installation Media

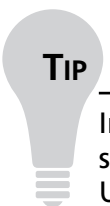
One of the more interesting features that Linux has over Windows is that it can be run from a variety of media. While windows is notoriously difficult to configure to run from a CD-ROM, there are Linux distributions that are capable of running off of a traditional hard disk install, CD-ROM, a Universal Serial Bus (USB) drive, or even a floppy disk. Each media type offers some security pros and cons, and not every distribution will be available on every media type. If you need the features of a specific distribution that doesn't come on the

media you prefer, you may need to make a compromise. You will need to research the different media options and choose one that fits in your environment. We will review some of the pros and cons of each.

Full Install

The *full intall* is the traditional install to a system's hard disk. Much like Windows, you boot up an install CD and walk through a guided install process. Most of the Linux distributions installed on the hard disk offer graphical user interface (GUI) install programs that walk you through the installation steps. There is no great advantage to using this type of distribution other than that the size of the hard disk allows you to install a lot of extra software. For a firewall, you generally want to keep the software running to a minimum to enhance security, so this shouldn't be a very big consideration. This type of installation also has the advantage that it will be easy to modify and alter the configuration if needed.

On the down side, this type of installation has all of the same disadvantages of a Windows bastion host. Namely that the entire system is sitting on the hard drive and if a hacker manages to compromise the root account, they will be able to install a virus or Trojan on the system that can survive future reboots. This type of install isn't any better or worse than if you were using Windows for your bastion host OS. Despite these concerns, this is the most common type of Linux firewall installation and most versions of Linux install the firewall components by default. This means if you download a version of Linux you like and install it to a hard disk, you will have a firewall waiting to be configured when you're done.



TIP

In the event that you discover your firewall has been compromised, it is considered best practice to wipe the system clean and rebuild it from scratch. Unfortunately, unless you have some means of isolating *all* changes that were made, you cannot ensure that it is safe to leave the system operational. One of a hacker's first steps is often to install a back door so that they can easily gain access to the device in the future. These backdoors include techniques such as modifying various systems commands so that detecting the back door is difficult. For this reason, rather than risk leaving a system operational that may be compromised, a complete format and reinstall is recommended.

CD-ROM

While you can get windows running off of a bootable CD-ROM or live CD, it takes a lot more work than it does with Linux. There are many versions of Linux designed specifically

to run from a CD-ROM, allowing you to turn virtually any machine into a firewall, router, or general-purpose PC. There is an obvious security advantage to having all of your configuration information on read-only media. Even if a hacker manages to compromise the system, all it takes is a reboot and it can be restored to its previous condition. The system can still fall victim to a virus or Trojan, but only until it is rebooted. Further, if the firewall system has a hardware failure such as a failed central processing unit (CPU), all you would need to do to restore your firewall would be to move the CD to a new system and reboot.

The primary advantage to a CD-ROM-based installation is also the primary disadvantage. If you burn the entire OS and configuration settings to a CD, any time you need to make adjustments you would need to burn a new CD-ROM. The cost of the CD media probably isn't an issue, but such a configuration may hinder your ability to remotely administer the system, which would be limited to making changes to the running configuration. Changes that remained after a reboot would require someone local to insert the CD-ROM containing the new configuration. If you needed to implement and test changes that required a reboot to take effect, this type of the setup would make things more difficult. Finally, due to simple space limitations on a CD-ROM, you may not be able to fit all of the needed software or functionality on a CD-ROM. That being said, if the firewall rules are relatively static and don't require frequent adjustment, a live CD could be a very attractive option.

USB Drive

If the space limitations are acceptable, a Linux-based firewall booting from a USB disk may offer the best compromise in security and flexibility. Having the operating systems and firewall software on a pen drive offers the same type of flexibility that a CD-ROM-based system provides, with increased storage capacity over that of a CD-ROM. If you purchase a USB disk that includes a physical write protect switch, you can make changes on the fly, like a live system, and then write protect the disk against modification when you are done. As the storage capacity of USB drive increases, you will be able to use a USB-based distribution that includes increasingly greater functionality. One key consideration with this type of media is that not all systems will support booting from a USB disk. While almost all newer systems support this option, many of the older systems that you may wish to install a free firewall on do not.

Floppy Disk

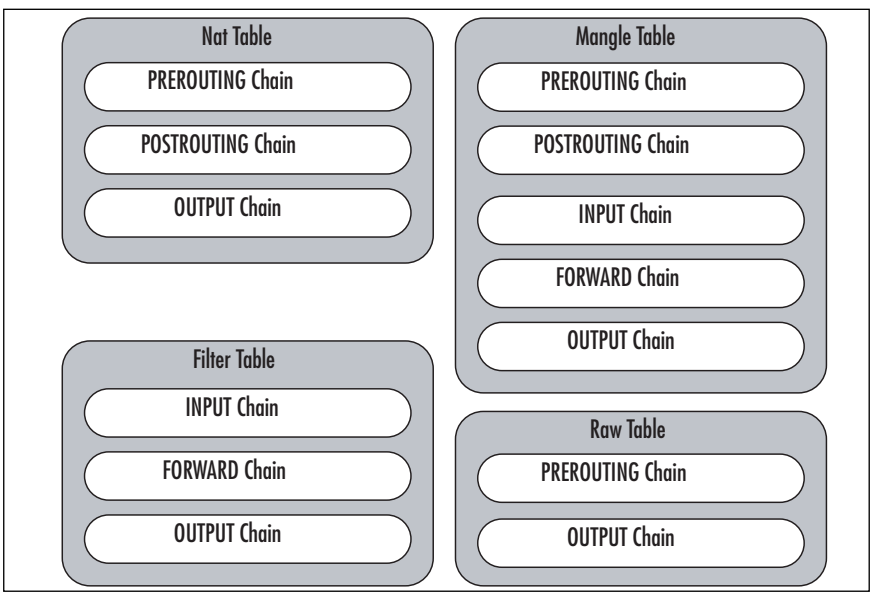
Although the functionality is typically very limited, there are many versions of Linux that can fit on a 3.5" floppy disk. The primary advantage of these distributions is their low resource requirements. Often, these systems only require 8 or 16 megabytes of memory and a 486 processor to function. The ability to toggle the write protect switch on the floppy can also provide a high degree of configuration flexibility and security. Considering the unreliable nature of floppy disks, it probably wouldn't be appropriate for use if an outage cannot

be tolerated. At the very least you should have duplicate floppy disks available in the event of a failure. Another disadvantage to these is functionality. Generally, these floppy-based distributions are single-purpose devices and lack much in the way of functionality. Another consideration is that due to the space restrictions on a floppy disk, these floppy-based distributions are almost always command line only, with no GUI for configuration or management.

Linux Firewall Operation

Before diving into the specific commands used to configure the Linux firewall, we will cover some basic Linux firewall vocabulary and how the firewall operates. netfilter contains the firewall logic, and iptables is the program that is used to modify the rules that the firewall uses. (See the netfilter home page at www.netfilter.org/.) These rules (or ACLs) define the rules used to permit or deny packets and how to react to denied packets. The current iptables use both tables and chains. *Tables* are the blocks of processing where various actions are performed on the packets. Different tables process different chains. *Chains* are a set of rules (or ACLs). There are four built-in tables: *nat*, *mangle*, *filter*, and *raw*, each of which processes different chains (see Figure 9.4).

Figure 9.4 netfilter Tables and Chains



The following tables and chains are not listed in any particular order, as a given packet may be impacted by multiple tables and chains as it is processed. The primary built-in chains are INPUT, OUTPUT, and FORWARD. In addition to these, you can create your own

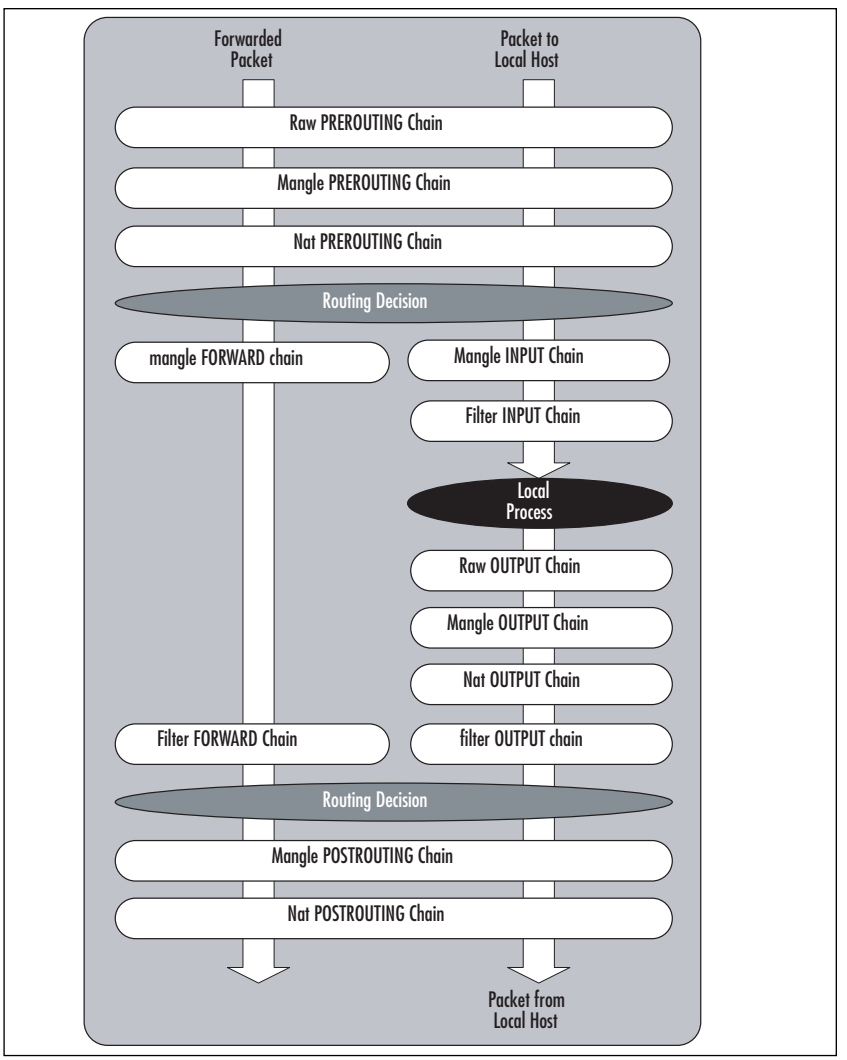
user-defined chains. Capitalizing the names of the chains is a common convention, but is not required.

A brief summary of the roles the tables and chains play is included for reference.

- **Nat Table** This table is referenced with a packet that is used to create a new connection.
  - **PREROUTING** This chain is processed as soon as a packet is received and before any routing decisions are made.
  - **POSTROUTING** This chain is processed before a packet is sent to an interface but after any routing decisions have been made.
  - **OUTPUT** This chain is processed for packets generated locally.
- **Filter Table** This is the default table that is used when the iptables command is used to modify the rules and do not specify an alternate table. This is where the bulk of a firewall's processing is consumed.
  - **INPUT** This chain is processed for packets destined for the local system.
  - **FORWARD** This chain is processed for packets passing through the local system.
  - **OUTPUT** This chain is processed for packets generated by the local system.
- **Mangle Table** This table is used for any specialized packet alterations that are needed. Examples are performing Network Address Translation (NAT) or manipulating various bits within the packet.
  - **PREROUTING** This chain is processed on incoming packets before a routing decision is made.
  - **POSTROUTING** This chain is processed last before a packet is sent to an interface.
  - **OUTPUT** This chain is processed before a routing decision is made for packets generated locally.
  - **INPUT** This chain is processed for packets destined for the local system.
  - **FORWARD** This chain is processed for packets passing through the local system.
- **Raw Table** This table is primarily used for packets that are exempt from connection tracking, and if required, are called before any other netfilter table.
  - **PREROUTING** This chain is processed as soon as a packet is received.
  - **OUTPUT** This chain is processed for packets generated locally.

After you have reviewed all the various tables and chains, it's worth discussing the overall packet flow. The key to remember is that not all packets traverse all chains. To further muddy the waters, packets will traverse different chains depending on whether they are sourced from the netfilter host, destined for the netfilter host, or just passing through the netfilter host. Remember this will save you time when troubleshooting your firewall rules in the future. Refer to Figure 9.5 for a diagram depicting the packet flow through netfilter.

Figure 9.5 Netfilter Packet Flow



Targets are the actions that should be taken when a packet matches a given rule. A target is specified using the *–j <target>* syntax (for jump). The primary targets used for a firewall are ACCEPT and DENY.

- **ACCEPT** The packet is accepted and processed by the rest of the TCP/IP stack.
- **DENY** The packet is dropped and no notice is given to the sender. While this does not honor the TCP/IP protocol specifications, it is considered the most secure option, because it denies a hacker useful information about the firewall. This behavior also has a negative side effect, which is if a system is trying to initiate a connection to a port that is blocked by a firewall, the connection attempt must time out before the initiating host gives up. If you use REJECT, the Internet Control Message Protocol (ICMP) port will allow the initiating system to abort the connection attempt immediately.
- **LOG** This allows you to perform kernel logging, which appears in the syslog log. Further options allow you to specify the log level and a descriptive prefix for the log entry.
- **RETURN** Processing continues in the previous chain at the rule just after the last rule processed in that chain.
- **QUEUE** This is a special target that will hold (or queue) a packet for processing by a userspace process.

Unlike some firewalls, netfilter allows you to apply multiple rulesets (chains) to the same interface. Although it may seem minor, this option creates a lot of powerful possibilities. For example, suppose you have an ACL and you want to permit all packets originating on the 192.168.1.0 network except those from 192.168.1.11, which is a host that a third-party uses and is not a completely trusted system. You want packets sourced from 192.168.1.11 with a destination port of 22, 25, 53, 80, and 443 to be permitted, while all other packets are blocked. (see Figure 9.6).

Figure 9.6 Cisco ACL

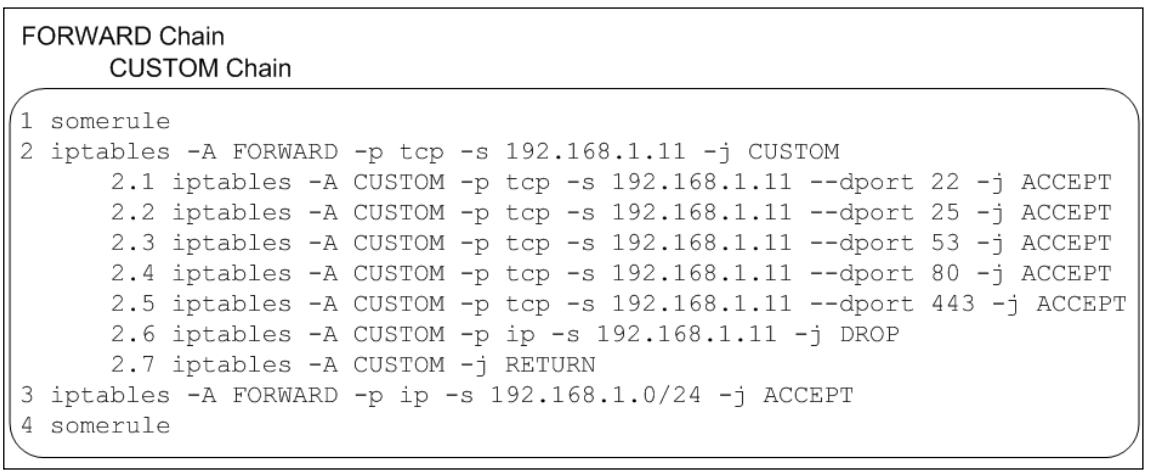
```
1 somerule
2 access-list 100 permit tcp host 192.168.1.11 any eq 22
3 access-list 100 permit tcp host 192.168.1.11 any eq 25
4 access-list 100 permit tcp host 192.168.1.11 any eq 53
5 access-list 100 permit tcp host 192.168.1.11 any eq 80
6 access-list 100 permit tcp host 192.168.1.11 any eq 443
7 access-list 100 deny ip host 192.168.1.11 any any
8 access-list 100 permit ip 192.168.1.0 255.255.255.0 any
9 somerule
```

In Figure 9.5, each line of the ACL is numbered for easy reference. The order of the rules is critical for proper operation of the firewall. Cisco processes each line in the ACL and

compares the rule with the packet in question. If it finds a match, it performs the indicated action and then stops any further processing of the ACL. This means if you reversed the order of rules 7 and 8, all packets from 192.168.1.11 would be permitted. This type of arrangement also means that a packet with a source IP address of 192.168.1.22 has to be compared against rules 2–7 before being accepted by rule # 8. With seven rules this will happen quickly, but if the ACL is lengthy this extra overhead could be CPU-intensive.

netfilter's ability to move through multiple chains for the same packet allows you to design your chains for greater efficiency (see Figure 9.7).

Figure 9.7 netfilter Chains



Using netfilter and iptables, you created rule # 2, which says that the source address is 192.168.1.11 for processing the CUSTOM chain. You can create the CUSTOM chain with the *iptables –N CUSTOM* command. Within the CUSTOM chain, you check for the five permitted destination ports (rules 2.1–2.5) and then reject everything else (rule 2.6). Rule # 2.7 has no matching criteria and will therefore match on any packet and instruct the packet to return to the FORWARD chain where processing can continue. FORWARD chain rule # 3 permits all other packets from the 192.168.1.0/24 network. This means that packets not sourced from 192.168.1.11 only have to be checked against rule # 2 and can then move through the chain(s) instead of being checked against all the rules. Figure 9.6 shows the flow of a packet through the rules. The actual rules as they would appear in iptables can be seen with the ***iptables –L*** command.

```
# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
```

```
target     prot opt source               destination
CUSTOM     tcp  --  192.168.1.11         anywhere
ACCEPT     tcp  --  192.168.1.0/24       anywhere

Chain OUTPUT (policy DROP)
target     prot opt source               destination

Chain CUSTOM (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  192.168.1.11         anywhere            tcp dpt:ssh
ACCEPT     tcp  --  192.168.1.11         anywhere            tcp dpt:smtp
ACCEPT     tcp  --  192.168.1.11         anywhere            tcp dpt:domain
ACCEPT     tcp  --  192.168.1.11         anywhere            tcp dpt:http
ACCEPT     tcp  --  192.168.1.11         anywhere            tcp dpt:https
DROP       all  --  192.168.1.11         anywhere
RETURN     all  --  anywhere             anywhere
```

Another advantage is that because rule # 2 sent you to another chain, you can make certain assumptions that you wouldn't otherwise be able to. For example, in the CUSTOM chain you could replace

```
iptables -A CUSTOM -p tcp -s 192.168.1.11 --dport 22 -j ACCEPT
```

with

```
iptables -A CUSTOM --dport 22 -j ACCEPT.
```

This is because the packet would not be in the CUSTOM chain without matching the *–p tcp* and *–s* 192.168.1.11 (source IP address). If you want to tweak the CUSTOM chain even more, the RETURN target in rule # 2.7 isn't strictly required. If the packet reaches the end of a user-defined chain without having a match, it will RETURN to the previous chain by default. If a packet reaches the end of a built-in chain without a match, it will use the policy target (typically DROP). Now that you have a feel for the flexibility and power of iptables and netfilter, let's look at some practical configuration examples.

Configuration Examples

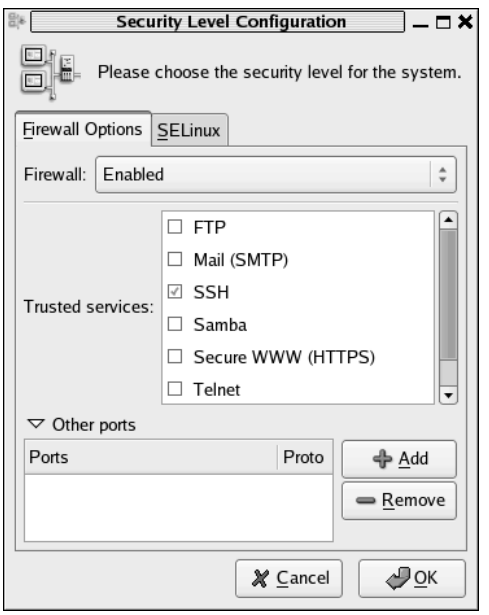
The next step is to demonstrate how to configure the netfilter firewall. This is a critical step, and the firewall should only be installed and configured after the underlying OS has been installed, updated, and hardened. These instructions assume you are working with an otherwise secure system and now need to configure the firewall functionality.

To make sure the firewall is enabled, you can run *chkconfig —list*, which lists all of the services and the run levels they are configured to start in. For example, you get the following output:

```
chkconfig --list | grep iptables
iptables    0:off    1:off    2:on    3:on    4:on    5:on    6:off
```

This output tells you that iptables will start in run levels 2–5. You can set it to run in run levels 2–5 by using the **chkconfig –level 2345 iptables on** command. If you are using a GUI window manager, you probably have another graphical application to see this information. For example, in Fedora Core 5, you can navigate to **System | Administration | Security Level and Firewall**, which opens the screen shown in Figure 9.8.

Figure 9.8 Fedora Core Firewall GUI



You can enable or disable the firewall by going to the **Firewall Options** tab and selecting **Enabled** or **Disabled**. This particular interface in Fedora Core 5 also allows you to perform limited configurations of the firewall rules (e.g., by checking the Trusted Service SSH, a rule would be added to allow inbound connections on TCP port 22). Because any graphical interface provided will likely vary from one distribution to another, we use the command line to configure the firewall.

Deleting Rules and Chains

With many Linux distributions, the netfilter firewall will become enabled, but with an empty ruleset. In others, it might come with the firewall enabled and a very liberal ruleset in place. Let's start configuring a Linux firewall by deleting any default rules that are present. You can use *iptables –L* (or *—list*) to list the current rules. An empty default ruleset should look like this:

```
iptables -L
Chain INPUT (policy ACCEPT)
Target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
Target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
Target     prot opt source               destination
```

If there are any default rules present, they can be deleted using the *iptables –F* command. The *–F* option means to flush, which is equivalent to using –flush. This will clear all rules out of any existing chains. If distribution has any additional chains created beyond the default, you can delete a custom chain by using the *iptables –X customchain* command. Creating your own user-defined chain is accomplished using the *iptables –N customchain* command. In addition to the individual rules within a chain, the built-in chains have a default policy associated with them. This policy tells netfilter what to do if a packet reaches the end of the chain without finding a match. While the default policy is to ACCEPT, it is better to change this to DROP by using the *–P* option, which sets the default policy for that chain, as follows:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

Permitting Traffic to and from the Firewall

Now that you have a clean slate and a default policy of DENY, the first thing you will want to do is make sure that management traffic is permitted to the firewall itself. This is done first, because once you have enabled the firewall with a default policy of DENY, you will not be able to manage the firewall remotely until you have configured the firewall rules to permit the management traffic. This traffic is processed against the INPUT chain, because the destination is the netfilter host itself. To allow secure shell (SSH) connections to the firewall, use the following command:

```
iptables -A INPUT -p tcp -s 192.168.99.0/24 --dport 22 -j ACCEPT
```

In this example, you are appending (*-A*) a rule to the INPUT chain to allow traffic from the 192.168.99.0/24 network to a destination port of TCP 22. With no other configurations, all other traffic through or to the firewall would be dropped. This will show up in the rule listing as follows*:*

```
iptables -L INPUT
Chain INPUT (policy DROP)
```

```
Target     prot opt source               destination
ACCEPT     tcp  --  192.168.99.0/24      anywhere            tcp dpt:ssh
```

Although the aforementioned rules will permit the inbound SSH session, there is currently no rule to permit the reply traffic for the SSH session. If you were to change the default policy for the OUTPUT chain to ACCEPT, this would permit the reply packet, but we will instead address this more securely in the next few examples.

If you also wanted to allow 192.168.99.99 access to the firewall with a destination of TCP port 80, you could use the same syntax with *–A* to append the rule, which would put the new rule for port 80 *after* the rule for port 22. You could also use *–I* for *insert*, as in the *iptables –I INPUT 1* **–p tcp –***s 192.168.99.99 -–dport 80 –j ACCEPT* command. This would insert the new rule in the INPUT chain as rule # 1, meaning the rule for port 80 would come *before* the rule for port 22. Remember, this is still permitting only half of the conversation; you still need to permit the outbound reply packets. It is sometimes useful to list the chains with rule numbers using the *iptables –L —line-numbers* command.

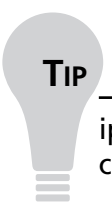
For outbound traffic (i.e., traffic generated by the firewall), you need to create rules in the OUTPUT chain. To enable syslog traffic from the firewall to a remote syslog server (192.168.1.99), you would enter the following*:*

```
iptables -A OUTPUT -p udp -d 192.168.1.99 --dport 514
```

This assumes you are using the default UDP syslog port of 514. Because syslog over UDP is a one-way conversation, you will not need to permit any inbound replies to the syslog traffic. The OUTPUT chain is where you need to permit replies for permitted traffic that you allowed inbound in the preceding examples. You could create rules to permit SSH and HTTP specifically, but there is also a way to permit all traffic that is a reply to a permitted session. You can enter

```
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

This will instruct netfilter to permit any outbound traffic that is part of an established session (ESTABLISHED). The RELATED keywork is similar, but is for traffic that is part of a different sessions, but where the sessions is related to an established session. Some protocols will open additional ports (such as FTP) as part of their normal behavior. For those that netfilter understands, it can see the request for the additional port and permit that new session.



TIP

iptables commands that manipulate the chains or rules themselves use uppercase letters:

```
-A append, -D delete rule, -I insert, -R replace, -L list, -F flush, -N
new, -X delete chain
```

Lowercase options are used for specifying rule parameters*:*

```
   -s source address, -p protocol, -d destination address, -j jump, -i in-
interface, -o out-interface
```

Simulating the Windows Firewall

Now let's configure the firewall. The built-in firewall on Windows XP is enabled by default with service pack 2 or better. The standard configuration is to allow outbound connections from the host system, and deny inbound connections unless they are explicitly configured. The Windows firewall also allows any traffic that is a reply to traffic that the host originally generated outbound. After you execute the **iptables –F** command to flush out all of the previously configured rules, the following commands would configure the Linux host similarly:

```
iptables -P OUTPUT ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The —*state* extensions track the current status of the connections. By specifying ESTABLISHED or RELATED, the firewall allows packets that are part of a currently established session, or packets that are starting a new session, but where the session is related to an existing session (such as an FTP data session). If you were hosting a service on this system, such as a Web server, you would need to configure the INPUT chain appropriately. This configuration would afford any Linux system a minimum level of firewall security with virtually no impact to it's overall functionality.

Simulating a Home Network Router

With the basics of iptables configuration out of the way, let's tackle a more practical example. For a typical firewall, there is very little traffic destined *to* or *from* the firewall itself. In general, the only traffic that would fit this profile would be administrative sessions to configure the firewall itself. The vast majority of a firewall's traffic is passing through the firewall, and will thus be checked against the FORWARD chain. The following examples would configure the Linux firewall with the same access controls as a typical home network router such as a Linksys or Netgear router/firewall. This example assumes that 192.168.1.0/24 is the internal network on interface *eth0* and the external interface is *eth1*.

```
iptables -P OUTPUT ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -A INPUT -p tcp -s 192.168.1.0/24 -i eth0 --dport 80 -j ACCEPT
```

```
iptables –A FORWARD –s 192.168.1.0/24 –i eth0 –o eth1 –j ACCEPT

iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

NOTE

Always remember that if you have configured the default policy for a chain to DROP (for example, iptables -P FORWARD DROP) that you will need to include an explicit rule to permit the return traffic. This can be done by using the following command:

```
iptables -A <CHAIN> -m state --state ESTABLISHED,RELATED -j ACCEPT
```

So if you wanted to permit the return traffic for a FORWARD chain, you would enter

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

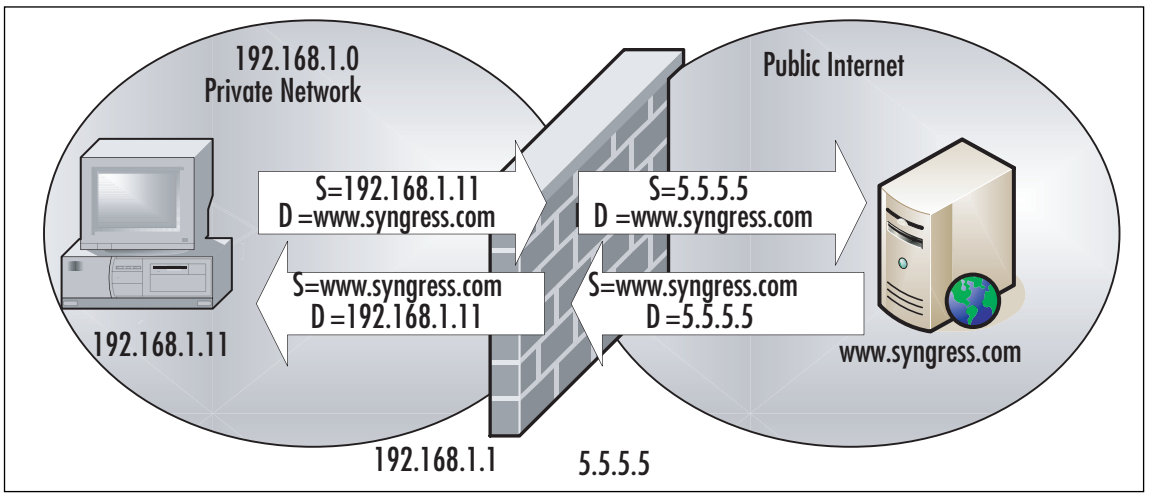
Many hours of troubleshooting Linux firewalls have been spent by overlooking a rule that permits the return traffic.

The INPUT chain allows port 80 to go to the firewall itself from the internal network. Many the home routers have a Web interface for configuring them, and while your configuration may not need this port open to the firewall, it is included here to help emphasize how the different chains are used. It is important to specify the input interface (using *–i*) so that the source IP cannot be spoofed by an external attacker. In this way, you ensure that even if a packet was generated with the proper source IP, if it came in on the outside interface (*eth1*) it would not match the rule and would thus not be permitted. The FORWARD rule allows any outbound traffic from the internal network to the external network. This configuration is simple to implement; however, the 192.168.1.0 IP range is a private IP range and is not routable on the Internet. Thus, this range wouldn't allow traffic from the internal network to the Internet quite yet. To make this Linux firewall a useful replacement for a home network router, you need to enable NAT, which allows all of the systems on your internal network to appear as a single IP address when communicating on the Internet.

Let's review NAT in it's various incarnations. In principle NAT is simple, but in a complex environment it can get confusing. As always, good documentation can help keep things straight. Basically, NAT means that the NAT device (in this case the Linux netfilter firewall) will change the IP address in a packet and retransmit that packet. Depending on your needs, you can alter the source IP address (source NAT [SNAT]), the destination IP address (destination NAT [DNAT]), or both (double NAT). For example, take a home router. The objective behind the NAT capability is to allow all of the internal hosts to communicate on the Internet using the single public IP provided by your Internet Service Provider (ISP). (In this

case, SNAT is being used.) As each of the hosts on your private network make a connection to an Internet server, the firewall is altering the source address to look like the public IP from your ISP. By doing this, the return traffic can find it's way back to the firewall and be retranslated and sent to the originating host (see Figure 9.9).

Figure 9.9 SNAT



In Figure 9.9, the internal host has a private IP address of 192.168.1.11. The public address of the firewall is 5.5.5.5, which is provided by the ISP. If a host on the private network wants to make a connection to www.syngress.com using a Web browser, the connection is sent with source address 192.168.1.1 to a destination address of www.syngress.com. The firewall alters the source address to it's own public IP address of 5.5.5.5 and sends the packet on its way. When the server replies to destination 5.5.5.5, the firewall again edits the packet, this time inserting a new destination of 192.168.1.11. All of this takes place and is transparent to the 192.168.1.11 host and the www.syngress.com server. When multiple hosts are using SNAT, the firewall tracks which connections belong to which private hosts using the port numbers. While the destination port of the Web server remains static (typically port 80 for the Web), the source port is usually a random port above 1024. By tracking the source port, the firewall knows which address belongs to which session. In the event that two hosts attempt to use the same source port, the NAT device edits the source port of one of the connections and replaces it with another random source port. When the return traffic is received, it translates the source port back, just like it did for the IP address. Because this method of NAT relies heavily on using the source port number, it is sometimes referred to as port NAT (PNAT).

To add the SNAT functionality to the example firewall, use the following command:

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 5.5.5.5
```

The *–t* option is used to specify the table we want to modify, and the *–A* option specifies that we are going to append this rule to the *POSTROUTING* chain. By specifying the outbound interface, we are ensuring that the SNAT only occurs as traffic leaves the private network, meaning only in the proper direction.

NOTE

SNAT can only be performed in the *nat* table. However, the rules for SNAT can only go in the *POSTROUTING* chain of the *nat* table. This means that any time you use SNAT, your rule will contain *–t nat –A POSTROUTING.*

The jump target SNAT is self explanatory. The *—to-source* option specifies what IP address we want to use as the new source address. SNAT assumes we have a static IP address to SNAT the outgoing packets to. While this is likely the case in a corporate environment, a more appropriate solution to more closely mimic the configuration of a home router would be to use the *MASQUERADE* command:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

The masquerade command does not require an IP specification, and will use the IP address of the firewall interface. You might be wondering why you wouldn't use the masquerade target all of the time instead of the SNAT target. Because the source IP is static, the SNAT target will cause the NAT calculations to be performed once for a given session. Subsequent packets belonging to that session are handled the same way as the first. With the masquerade target, each packet is checked for the source IP to use, which requires more overhead than with SNAT. This is why SNAT is preferable if you have a static source IP address, and masquerade is your only option if you do not have a static source IP address to use.

Additional Commands

By this point, you should have a relatively solid grasp of how to configure a Linux firewall. So far we have covered all of the core commands to permit and deny the traffic. Another useful command for your Linux firewall deals with logging packets. If you want to log everything passing through the firewall, use the **iptables –A FORWARD –j LOG** command. While simple, this would likely generate an excessive amount of logging traffic. You also might want some additional control of how the logging occurs. There are some additional options to provide this functionality. Of particular note are the *—log-level* and *—log-prefix* options.

The *—log-level* option allows you to specify what logging level is used for the LOG rule. The effect this log level has depends on how you have your kernel logging configured (via syslog or syslog-ng). When you combine the custom logging level of iptables with the syslog

configuration, you can have syslog act in any manner of ways based on the firewall logs, including sending e-mails for certain events. The *—log-prefix* option allows you to insert up to a 29-letter string in front of the log entry. This can be useful for troubleshooting purposes. Some examples of information you could place in log prefix would be the name of the chain that generated the log entry such as *iptables –A FORWARD –j LOG —log-prefix "from FORWARD chain."*

NOTE

While a packet that matches an ACCEPT, DENY, or DROP rule will stop traversing any other chains, this is not true of packets that match a log rule. After matching the log rule, the packets continue through any appropriate chains to be processed. Keep this in mind, so that you can configure an additional rule and action for the packet if desired.

Now that you can create a working ruleset for netfilter, you will want to save it. There are two commands of note: one for saving the configurations and one for loading a saved configuration. You can use the **iptables-save** command to generate output that is the current active ruleset. By default, it will only generate the output to the stdout, meaning it will display in the console. To save this output, redirect it to a file. To redirect the current ruleset to a file called */etc/ruleset*, you would type **iptables-save > /etc/ruleset**. If you want to save the current packet counts and rule counts, use the **iptables-save –c > /etc/ruleset** command. Individual tables can be saved separately by specifying the *–t* option using the **iptables-save –t mangle > /etc/ruleset** command.

Restoring a ruleset is accomplished using the **iptables-restore** command. Like **iptables-save**, the restore function takes only two optional arguments. The *–c* option will cause iptables to load the saved packet and byte counts, overwriting the current count values. The default behavior when using *iptables-restore* is to flush the ruleset before loading the saved ruleset, thus all previous rules are lost. If you wish to override this behavior, you can use the **–n** option, in which case the rules will be added to the existing ruleset, and will only overwrite if there is a duplicate rule. You can use the **iptables-restore < /etc/ruleset** command to pipe the saved configuration to iptables-restore.

Command Summary

The following is a brief summary of the most useful iptables commands for easy reference, along with some examples to make the command usage more clear. Bear in mind this is not an exhaustive list of commands; it only represents the most important commands for configuring your firewall. For a complete list, refer to the iptables man page.

- *–A* appends a rule to a chain. *iptables –A INPUT –p icmp –j ACCEPT* will add the rule to permit ICMP at the bottom of the *INPUT* chain in the *FILTER* table.
- *–D* deletes a rule from a chain. *iptables –D INPUT –p icmp –j ACCEPT* will delete the matching rule from the *INPUT* chain. *iptables –D INPUT 3* will delete the third rule from the top in the *INPUT* chain.
- *–I* inserts a rule in a chain. *iptables –I INPUT 5 –p icmp –j ACCEPT* will insert this rule as the fifth rule in the *INPUT* chain
- *–R* replaces a rule in a chain. *iptables –R INPUT 4 –p icmp –j ACCEPT* will replace the fourth rule in the *INPUT* chain with this new rule.
- *–L* lists the rules. *iptables –L* will list all rules and *iptables –L INPUT* will list all rules in the *INPUT* chain only.

iptables –t nat –L

would list all the rules in the nat table only.

- *–F* will flush (delete) the rules. *iptables –F* will delete all rules in all chains. It will not delete chains, only the rules inside the chains.
- *–Z* will zero the packet and byte counters. *iptables –Z* will delete all of the counters. *iptables –Z FORWARD* will delete all of the counters in the *FORWARD* chain only.
- *–N* will create a new chain. *iptables –N CUSTOMCHAIN1* will create a new chain named *CUSTOMCHAIN1*.
- *–X* will delete a chain. *iptables –X CUSTOMCHAIN1* will delete the custom chain named *CUSTOMCHAIN1*.
- *–P* will change the policy for a chain. *iptables –P INPUT ACCEPT* will change the policy for the *INPUT* chain to *ACCEPT*

The policy for a chain does not need to be limited to *ACCEPT* or *DENY*; it could use a custom chain for a target, if desired.

Option Summary

- *–p* specifies the protocol to match (works with "!"). *iptables –A FORWARD –p tcp* will add a rule to match any TCP packet to the FORWARD chain. *iptables –A FORWARD –p ! tcp* will match any packet that was not TCP.
- *–s* specifies the source address to match (works with *!*). *iptables –A FORWARD –s 192.168.1.99* will match any packet with a source address of 192.168.1.99. *iptables –A FORWARD –s ! 192.168.1.99* will match any packet that did not have a source address of 192.168.1.99.

- *–d* specifies the destination address to match (works with *!*). *iptables –A FORWARD –d 192.168.1.99* will match any packet with a destination address of 192.168.1.99.
- *–i* specifies the network interface that the traffic was received on (works with *!*). *iptables –A FORWARD –i eth0* will match any packet entering the eth0 interface.
- *–j* specifies the target. *iptables –A FORWARD –p tcp –j DENY* would create a rule at the bottom of the *FORWARD* chain that will *DENY* any TCP packet.
- *–o* specifies the network interface that the traffic was sent out of (works with "!"). *iptables –A FORWARD –o eth1* would match any packet leaving on the *eth1* interface.
- *–t* specifies the table to manipulate. *iptables –t nat –A POSTROUTING –p tcp –j DENY* will add a rule to the bottom of the *POSTROUTING* chain in the NAT table, to *DENY* any TCP packet.

If you don't specify the *–t* option, iptables assumes you are working with the filter table.

- *–v* specifies to be verbose. *iptables –L –v* lists all of the rules and includes packet counts per chain and per rule.
- *—line-numbers* specifies that the rule list should be numbered:

```
iptables -L --line-numbers
```

  This option makes it easier to know what number to use for the commands that take a rule number as an argument, such as *insert*, *delete*, *replace*, and so on.

- *-m* will match packets based on certain protocol-specific criteria. Because the match options are protocol specific, *-p* (*tcp/udp/icmp*) must be used with *–m*. Some common examples include:
  - *-m —sport* allows you to match packets based on the TCP or User Datagram Protocol (UDP) source port.
  - *-m —dport* allows you to match packets based on the TCP or UDP destination port.
  - *-m multiport* allows you to match packets based on multiple port numbers within the same rule. *iptables –A FORWARD –p tcp -m multiport —dport 22,25,53 –j DROP* would *DROP* any TCP packet with a destination port of 22, 25, or 53.
  - *-m state —state* will allow you to match packets based on the state of the connection. *iptables –A FORWARD –p tcp –m state —state NEW –j LOG* would *LOG* any TCP packets that were being used to initiate a new connection.

There are four recognized states*: NEW*, *ESTABLISHED*, *RELATED*, and *INVALID*

netfilter and iptables give you powerful packet filtering and manipulation capabilities for free. With Linux distributions available for free download, a firewall is within any company's reach. Because of this, deploying firewalls internally to protect highly sensitive systems or data is becoming increasingly viable. If you want to obtain a Linux firewall without having to install Linux, try any of the many live CDs that are available. Some excellent choices are be Knoppix or Slax.

GUIs

While the console commands that are used to manipulate and configure netfilter are not terribly complicated, they can sometimes get very lengthy. As the length of the command line grows, the chances of an accidental error increase. Alternatively, you may not like working on the command line, in which case there are a wide variety of GUI and menu-driven interfaces available for netfilter. In most cases, these menu-drive interfaces use your input to create the appropriate iptables commands, and alleviate you having to know the various switches and options to use. There are a large number of GUI interfaces available to configure your netfilter firewall, which are listed in the following section in approximate order of ease of use. All else being equal, I have demonstrated the GUIs that are available on a wide variety of platforms over an equal quality choice that only works with one distribution. In general, simpler also means less full featured, so be aware that if you are trying to create a complex ruleset, some GUIs may not have the needed functionality.

Security Level Configuration

You can start the iptables GUI provided with Red Hat-based Linux distributions by navigating to **System | Administration | Security Level and Firewall**. You can also call the program directly by running *system-config-securitylevel* from a terminal window. While the interface looks nice, it is limited in what it can configure. Basically, all you can do with this GUI is permit or deny certain ports. Fedora Core 5 configures the *INPUT* and *FORWARD* chains to jump to a custom chain named *RH-Firewall-1-INPUT*. There is no ability to differentiate between ports permitted in the *INPUT* chain or the *FORWARD* chain, because all rules configured through the GUI are applied to this custom chain.

Some services are pre-defined for you. Placing a check next to **SSH** and clicking **OK** and then **Yes** to commit the changes, would create the following rule in the *RH-Firewall-1-INPUT* chain:

```
iptables -A RH-Firewall-1-INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j
ACCEPT
```

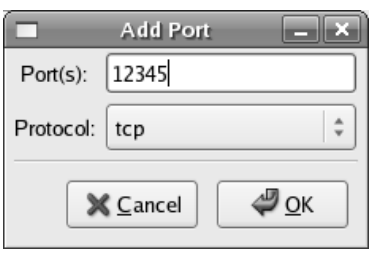
By expanding **Other ports** on the **Firewall Options** tab, you can enter a custom port number (see Figure 9.10.)

Figure 9.10 Custom Ports



Click **Add**, and enter the desired port number in the dialog box. Use the drop-down menu to select **TCP** or **UDP** for the protocol and click **OK** (see Figure 9.11).

Figure 9.11 Custom Port Dialog



This creates a rule identical to the SSH rule. There are no other configuration options. While this interface is adequate for a home PC that isn't running any services, it probably will not be adequate for a corporate firewall. If you need to configure access based on the interface in use or need to configure any NAT rules, you will need to use a different GUI. While you probably won't be needing this particular GUI as a corporate firewall, it is still useful to be familiar with it if you are running any Linux systems as workstations.

Lokkit

Lokkit is an ncurses-based menu for configuring your netfilter firewall. Lokkit is available for most major distributions and can be installed by default on some (such as Fedora Core 5). To start Lokkit, type **lokkit** in a terminal window. The first lokkit screen is shown in Figure 9.12.

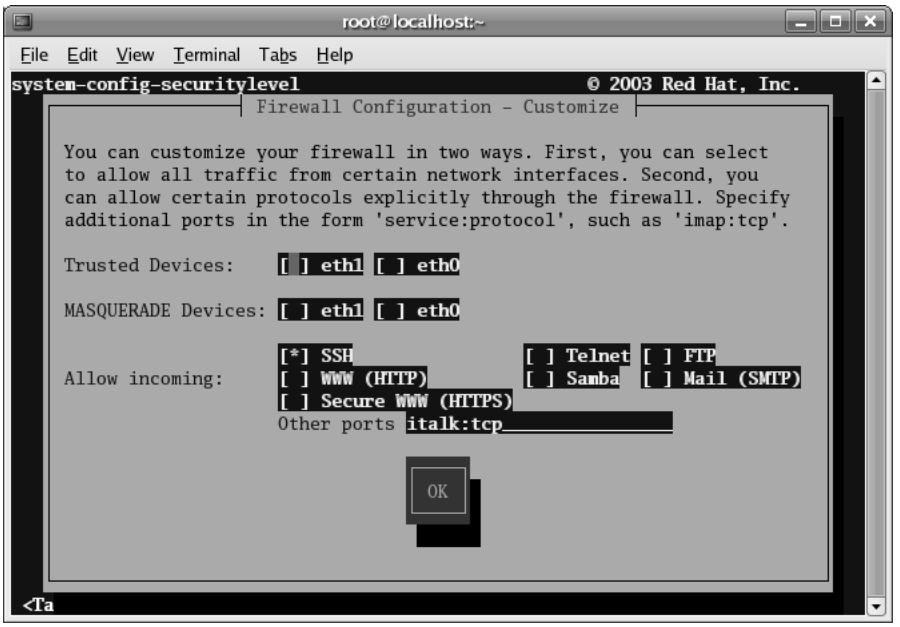
Figure 9.12 Lokkit Main Screen



You can navigate the menus using the Tab key and the space bar to toggle the equivalent of radio buttons, such as the **Enable** and **Disabled** options shown here. If you select **Enabled** on this screen, the default ruleset is applied. To edit any custom settings, press **Tab** until the **Customize** button is highlighted and then press **Enter**. The customization screen is shown in Figure 9.13.

Lokkit does provide a little more flexibility than the Security Level Configuration GUI discussed previously; however, it is still limited. By selecting an interface in Trusted Devices, all traffic from that interface will be permitted. This would typically be used to select the inside interface and designate it as trusted. You do have the option of enabling *MASQUERADE*. The interface you select is the one that will NAT outbound traffic, therefore, you would generally select your *external* interface. Some pre-defined services are available, and you can enter your own service information in the "Other ports" section. Once you are satisfied with your choices, press **OK** and then **Enter**. This will take you back to the main screen, where you press **OK** and then **Enter** to apply the changes.

Figure 9.13 Lokkit Customization Screen



If you attempt to configure an interface for *MASQUERADE*, it must also be marked as trusted or Lokkit will generate an error. Bear in mind that although *MASQUERADE* is limited, it has enough flexibility to configure a firewall similar to a typical home firewall/router device. This makes Lokkit a handy little utility to have in your repertoire should you need to configure a simple firewall quickly. The value of this utility is also increased, because it is available for a wide number of Linux distributions.

Firestarter

Firestarter is a GUI front end for netfilter and iptables, and its goal is to make it simple for the average user to configure their firewall and protect themselves. Firestarter runs on many Linux distributions and the installation is supported by many automated package management systems (such as *yum*, *apt-get*, and *portage*). Firestarter is an excellent choice if your needs are relatively simple for your firewall configuration. To install it manually, downloaded it from www.fs-security.com/download.php. Once it is installed, the first time you start the GUI interface you will need to perform some initial configuration. Follow these steps to configure firestarter:

1. Start the Firestarter GUI. In Fedora Core 5 this is done by navigating to **Applications | System Tools | Firestarter**. This will start the Firewall wizard. Click **Forward** on the **Welcome to Firestarter** screen.

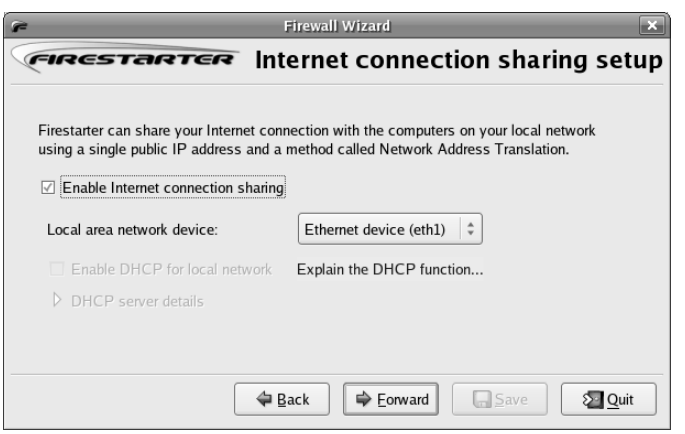
2. On the next screen, select your Internet-connected (i.e. external) network device from the "Detected device(s):" dropdown box (see Figure 9.14), and place a checkbox in the "IP address is assigned via DHCP" box. This is similar to the way a home router/firewall would be configured. When satisfied, click **Forward**.

Figure 9.14 Firestarter Network Device Setup



3. The next screen is the "Internet connection sharing setup" screen (see Figure 9.15), which is basically where you enable NAT. If you want to NAT all of the outbound packets to the external IP address, place a check in the "enable internet connection sharing" checkbox. When this checkbox is enabled, you can select the local area network device (i.e. the inside interface). If you only have two interfaces, it should be selected by default. When finished, press **Forward**.

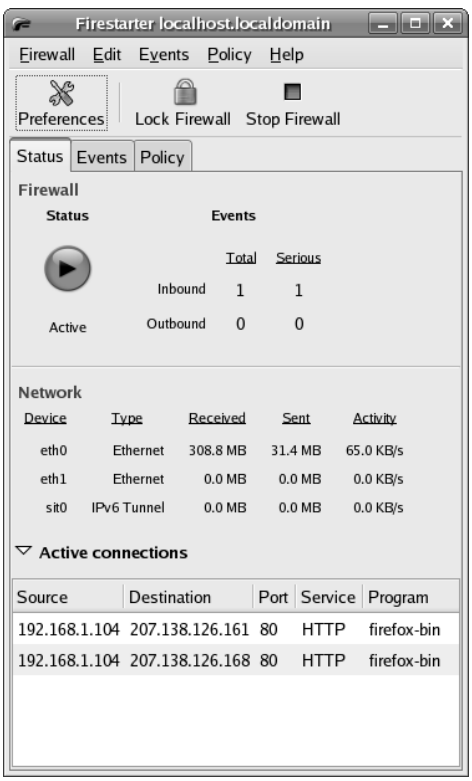
Figure 9.15 Firestarter Internet Connection Sharing Setup



4. On the final screen, leave the "Start firewall now" box checked and click **Save**. This will install a service to start Firestarter each time the system boots up. Firestarter will also change the default action for the chains to *DENY*; therefore, you must explicitly configure any ports you want to permit through the firewall.

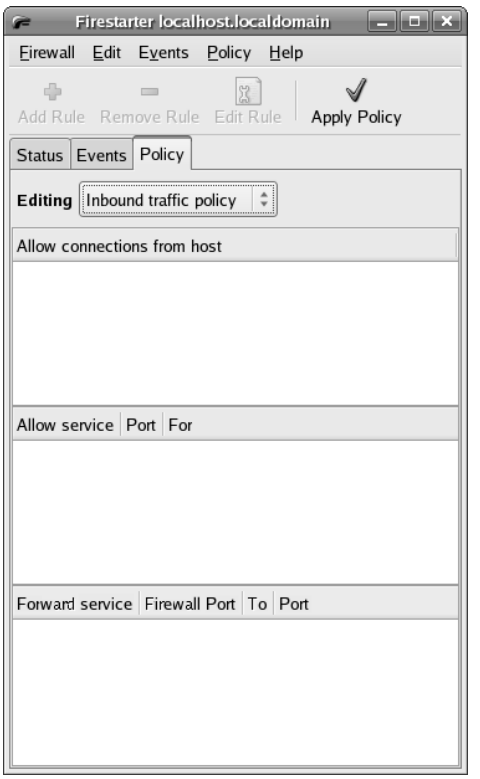
The main Firestarter GUI is shown in Figure 9.16. As you can see, it has a straightforward interface. The **Status** tab gives you high-level information such as sent and received data counters per interface, and a list of active connections. By clicking the **Stop Firewall** button, all of the iptables chains are flushed and the default action is changed to *ACCEPT*. This can be useful for troubleshooting issues to see if they are related to your firewall configuration.

Figure 9.16 Firestarter GUI



The "Events" tab lists recent blocked connection attempts. The "Policy" tab is where you configure certain rules to permit desired traffic (see Figure 9.17).

Figure 9.17 Firestarter Inbound Policy



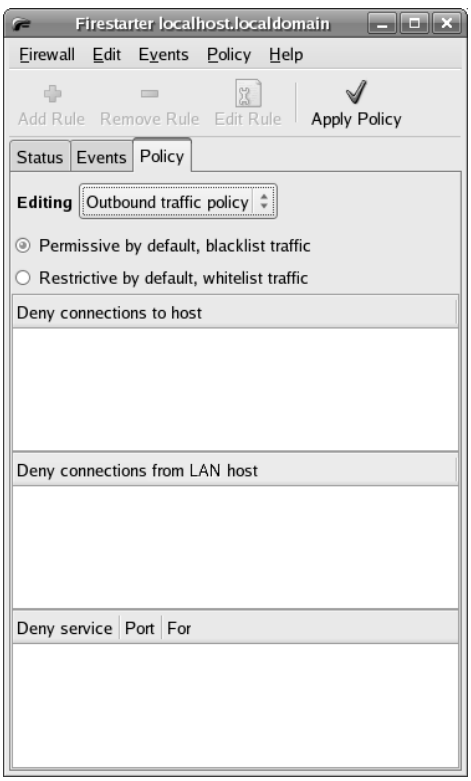
For example, if there was a Web server running on the Linux host, you could use the "Policy" tab to permit inbound connections to TCP port 80. The "Editing" dropdown box allows you to choose between inbound and outbound rules to edit. For the Web server example, we selected "Inbound traffic policy." The policy group you select when you click **Add Rule** determines where the policy is placed. The function of the various policy groups is outlined below.

- **Allow Connections From Host** This is used to configure a given IP address, hostname, or network. When you enter the IP information and create a rule in this policy group, all traffic from the configured source is permitted.
- **Allow Service** The allow service policy group is used to permit individual services. You can configure the source to be anyone including a specific IP, or network, or all local area network (LAN) clients. The LAN clients option permits the service through the firewall with a source address that is on the same subnet as the inside network adapter.

- **Forward Service** This option is used only when you are NATing. This allows the firewall to forward a specific port or range of ports, so that a service hosted on an internal NAT'ed device can receive inbound connections from the external network.

The "Outbound traffic policy" window shows a different set of policy groups (see Figure 9.18). There are also the additional radio buttons to select "Permissive by default," "blacklist traffic," or "Restrictive by default, whitelist traffic." If you select the permissive option (the default), all outbound connections will be allowed and any rules you create will be *DENY* rules. This is the same default behavior of most home firewalls. If you select the restrictive configuration, the default target for the table is *DENY*, and any rules you create will be *PERMIT* rules.

Figure 9.18 Firestarter Outbound Policy



The function of the different policy groups toggle between "allow" and "deny," based on whether you select restrictive or permissive mode. The policy groups are outlined here:

- **Allow/Deny Connections To Host** This policy group is used to globally permit or deny outbound access to a given host, IP address, or network range. This policy uses the destination to match the rule. You can use this policy group in permissive mode to list certain Web sites you do not want anyone to have access to.
- **Allow/Deny Connection from LAN Host** This policy group is used to permit or deny all access from a particular host, IP address, or network range. This policy uses the source to match the rule.
- **Allow/Deny Service** This policy group permits or denies traffic based on its destination port and source. When you are using permissive mode, this policy group can be used to block all access to the bittorrent ports. The traffic source can be anyone; the firewall itself, LAN clients, or an arbitrary IP, hostname, or network range.

Configuring the policies will satisfy the bulk of what you need to accomplish, but there are some additional configuration options available by navigating to **Edit | Preferences**. Selecting **Interface | Events** allows you to configure some useful options. The "Skip redundant entries" checkbox only makes one event entry for sequential event entries. This helps prevent the event windows from being flooded by repetitive alerts. You also have the option of entering certain hosts or ports as being exempt from triggering the event log. After making your selections, click **Accept**.

Another preferences setting of note is under **Firewall | Network Settings**. This allows you to enable Internet connection sharing (the same as during the initial wizard), and enable the firewall host as a Dynamic Host Configuration Protocol (DHCP) server. This allows you to configure the Linux host similarly to a home firewall, which generally acts as a DHCP server in addition to perform NAT and act as a firewall. The ICMP filtering window also allows you to filter ICMP packets. By default, the permit and deny rules configured by Firestarter apply to TCP and UDP, but not ICMP. This screen allows you to permit the desired types of ICMP traffic. Generally speaking, it is better not to allow any ICMP from the Internet to your firewall or internal network unless absolutely necessary.

One final setting you want to configure is under **Firewall | Advanced Options**. In the broadcast traffic section, check both options under **Broadcast traffic**. In general, you should not permit broadcast traffic to go through your firewall, as doing so poses a security risk. You also want to check the option to "Block traffic from reserved addresses on public interfaces," which is a common filtering tactic. Because the "private" addresses outlined in RFC1918 should not be routed through the Internet, there is never a reason to receive traffic sourced from any of those addresses on your outside interface. If you do, it is almost always a hacker attempting to bypass a poorly configured firewall.

Short of any advanced packet mangling, there isn't much you can't accomplish using Firestarter as your configuration tool. If you need to implement a more advanced configuration, use an alternate tool, or generate the configuration using Firestarter and use those chains as a starting point to add your own more advanced options.

Easy Firewall Generator

Easy Firewall Generator is not a GUI per se, but it does help simplify your netfilter configuration and avoid the need to be familiar with the iptables syntax. By using the Web page at http://easyfwgen.morizot.net/gen/index.php, you can enter the relevant information and click the **Generate Firewall** button. As you select options, if additional information is needed click the **Generate Firewall** button and the page will refresh and provide the additional input fields. When all of the required information has been entered, the page will change to a text page that can be copied and pasted for iptables to read as a saved configuration. On Fedora Core 5 the iptables configuration is stored in /etc/sysconfig/iptables. Although this method requires you to replace the default iptables configuration file used by your distribution, it is fairly painless, and supports all of the same basic functionality as Firestarter.

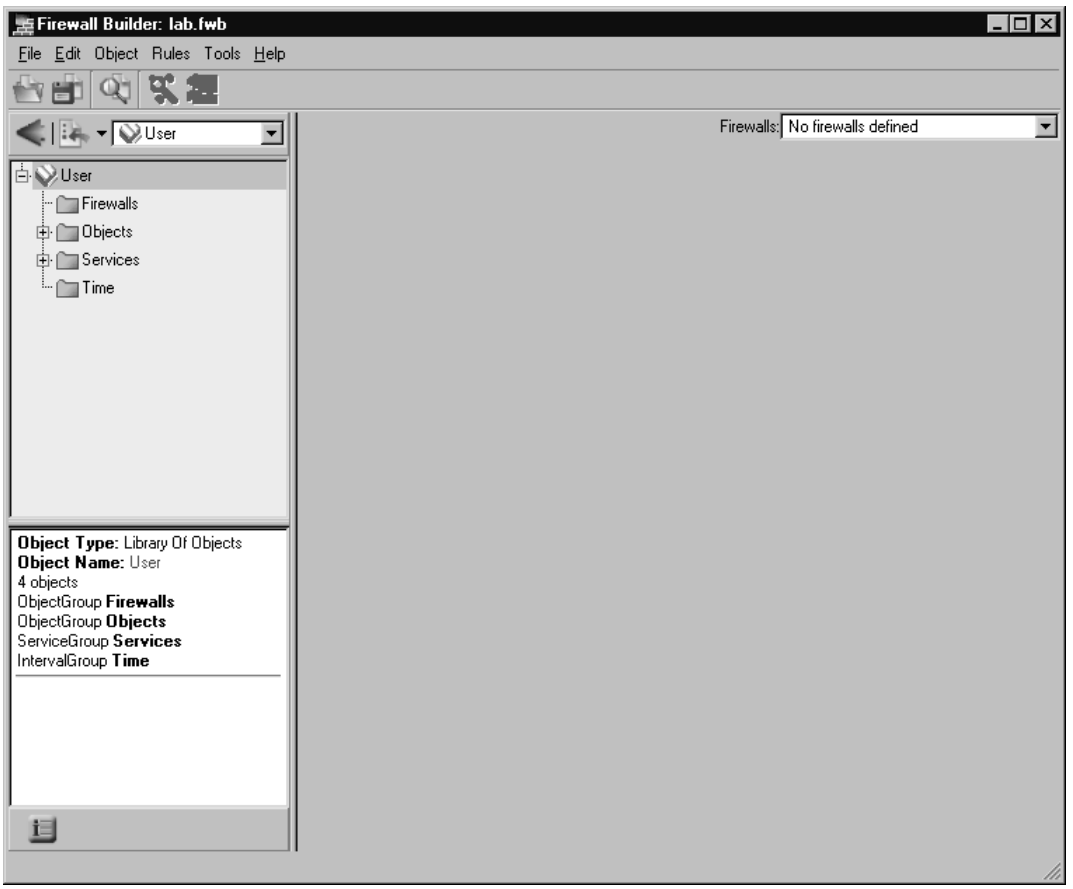
Firewall Builder

Firewall Builder is the most complete GUI offering for managing netfilter firewalls with features and capabilities comparable to some commercial firewall products. As is almost always the case, this functionality and capability come at a price: as far as netfilter GUI's are concerned, Firewall Builder is not the easiest to configure and use. If you want or need it's superior management capabilities, however, the extra effort is well worth it. (Download firewall builder from www.fwbuilder.org/.) Firewall Builder manages netfilter firewalls as well as ipfilter, OpenBSD PF, and (commercially) Cisco PIX firewalls. Firewall Builder runs on many popular operating systems including Red Hat, Mandrake, Suse, FreeBSD, Mac OS X, and Windows XP.

Firewall Builder operates differently than all of the GUIs covered so far. It uses an object-based approach. Essentially, you must define an object to represent any entity that you want to use in the firewall rules. In most cases this means a source, a destination, and a service (port) at a minimum. Both the configuration and the GUI bear a strong resemblance to that of the Checkpoint Firewall GUI. Once the objects are defined, you can drag and drop them into the rules in order to permit or deny communications between the two. For this example we use a Windows XP host to run Firewall Builder and configure a Linux netfilter firewall.

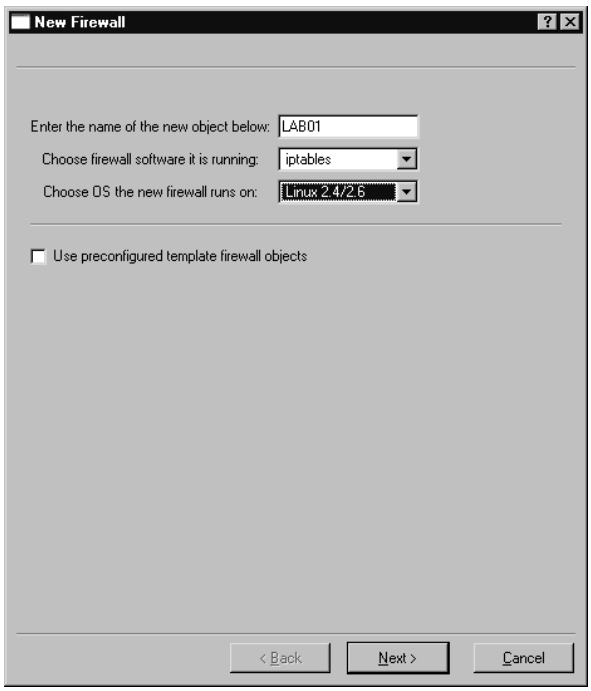
1. Install Firewall Builder.
2. Start the GUI by navigating to **Start | Programs | Firewall Builder 2.1 | FWBuilder**, which opens the main Firewall Builder window. It is divided up into an objects tree (the left pane) and the dialog area (the right pane) (see Figure 9.19).

Figure 9.19 Firewall Builder



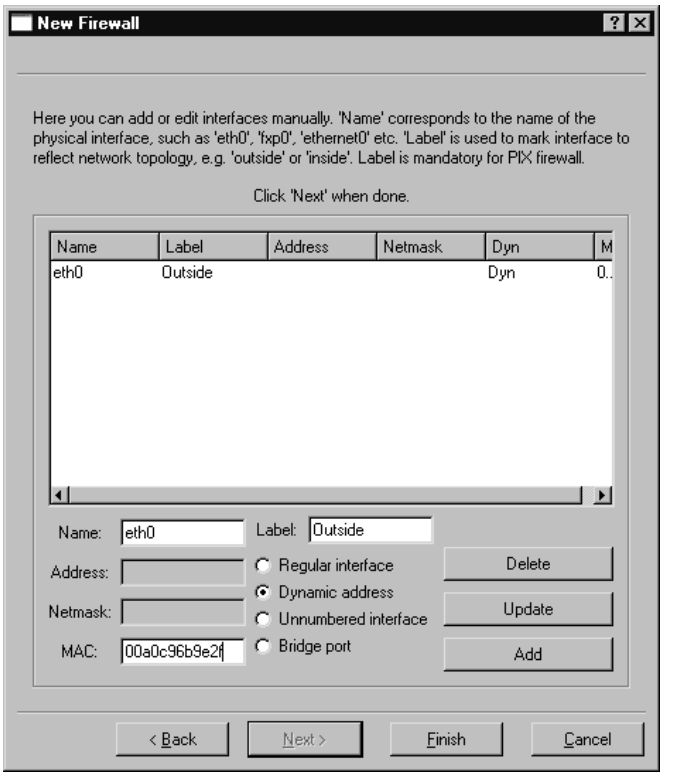
3. Initially, the dialog area will be empty. In order to add the first firewall (in this case a netfilter firewall) on the same host as you are running Firewall Builder, select **Firewalls** in the object tree.
4. Right-click and select **New Firewall**, which will open the New Firewall dialog box (see Figure 9.20).

Figure 9.20 FWBuilder New Firewall Wizard



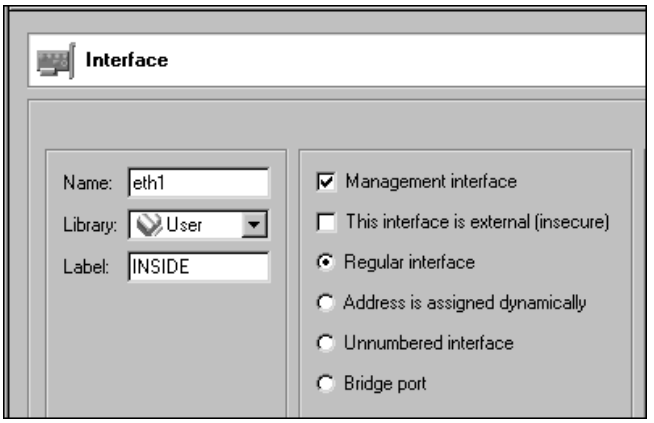
5. Enter the name for the new firewall (in this case LAB01).
6. For the firewall software, select **iptables**.
7. Choose **Linux 2.4/2.6** for the OS and click **Next**.
8. The next window allows you to configure the interfaces on the firewall. You can do so manually, or if the firewall is running SNMP, you can discover them via SNMP. We select **Configure interfaces manually** and click **Next**.
9. The manual interface configuration window is shown in Figure 29.21. Enter the relevant information for each network interface. The **name** must correspond to the actual interface name (same as if you entered ifconfig on the Linux host), such as *eth0*. The **Label** is a human friendly name for easy reference such as *OUTSIDE*. When you are done entering the information for a given interface click **Add**.

Figure 9.21 FWBuilder Manual Interface Configuration



10. When you have entered the information for all interfaces (typically an *INSIDE* and *OUTSIDE*), click **Finish**.
11. You must designate one of the interfaces on the firewall as the management interface, typically the *INSIDE* interface. Do this by navigating to the firewall in the object tree. As you select each interface in the object tree, there is a "Management interface" checkbox in the dialog area. Check this box for the interface you want to use. This will be the interface that FWBuilder uses to connect and upload the firewall rules to. The interface properties are shown in Figure 92.22.

Figure 9.22 Management Interface



Now that you have the basic firewall defined, you need to define something for it to talk to. In this case, let's assume that 192.168.1.0/24 is your internal network, and you want to allow outbound Web browsing and access to an internal Web server (WEB1). For starters, you need to create an object to represent the internal network. Follow these steps to create the network object.

13. Navigate to **Objects | Networks** in the object tree.
14. Right-click **Networks** and select **New Network**.
15. Enter **INTERNAL** for the name of the network, and use 192.168.1.0 for the Address field. Enter 255.255.255.0 for the Netmask and click **Apply**.
16. Let's go ahead and next create an internal Web server at 192.168.1.12. Right-click **Objects | Hosts** in the objects tree and select **New Host**.
17. Enter **WEB1** for the name of the object. Click the **Use preconfigured template host objects** check box and click **Next**.
18. Select **PC with one interface** and click **Finish**.
19. Expand the object tree to **User | Objects | Hosts | WEB1 | eth0 | WEB1**. Edit the IP address to be 192.168.1.12 and click **Apply**.
20. Next, define the appropriate services to allow Web browsing. Right-click **Services | TCP** and select **New Service**.
21. Enter **HTTP** for the name. Leave the source port ranges at zero, but change the destination port range to start and end at 80 and click **Apply**.
22. Repeat steps 20 and 21 for HTTPS on port 443 for secure Web pages.

This can be a lot of trouble; however, the real strength of an object-oriented approach is seen when it comes time to configure the rules. With all of the appropriate objects in place, let's define the rules to permit the inbound HTTP traffic.

23. In the top panel of the dialog area right-click and select Insert Rule.
24. Allow inbound HTTP to WEB1. Click on WEB1 in the object tree and drag it to the destination cell for rule 0.
25. Now drag the HTTP and HTTPS service from the object pane to the Service cell in rule 0.
26. Right-click the big red dot in the Action column and select Accept. This allows the inbound Web traffic to access WEB1.
27. To allow outbound Internet access, create another rule by right-clicking on rule zero and selecting Add Rule.
28. Drag and drop HTTP and HTTPS from the object tree into the Service column of rule one.
29. Drag the Network object **INTERNAL** from the object tree to the **Source** column of the new rule.
30. Right-click on the Action column for rule 1 and change the action to ACCEPT. Your policy should look like the one shown in Figure 9.23.

Figure 9.23 Sample FWBuilder Policy

Policy | NAT | Routing

	Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0	Any	WEB1	TCP HTTPS TCP HTTP	All			Any		Inbound Web
1	INTERNAL	Any	TCP HTTP TCP HTTPS	All			Any		Outbound Web

31. Although our rules seem simple at the moment, let's apply them to see how things work. First, save your work by navigating to **File | Save** or **File | Save As**.
32. Next, right-click the **LAB01 Firewall** and select **Compile**.
33. When the "Select Firewalls for compilation" window comes up, **LAB01** should be checked. When satisfied with your selection, click **Next**. When the compilation is complete you should see "Success" in the "Progress" column. After verifying that the compilation was successful, click **Finish**.

Tools & Traps…

Don't Block Yourself

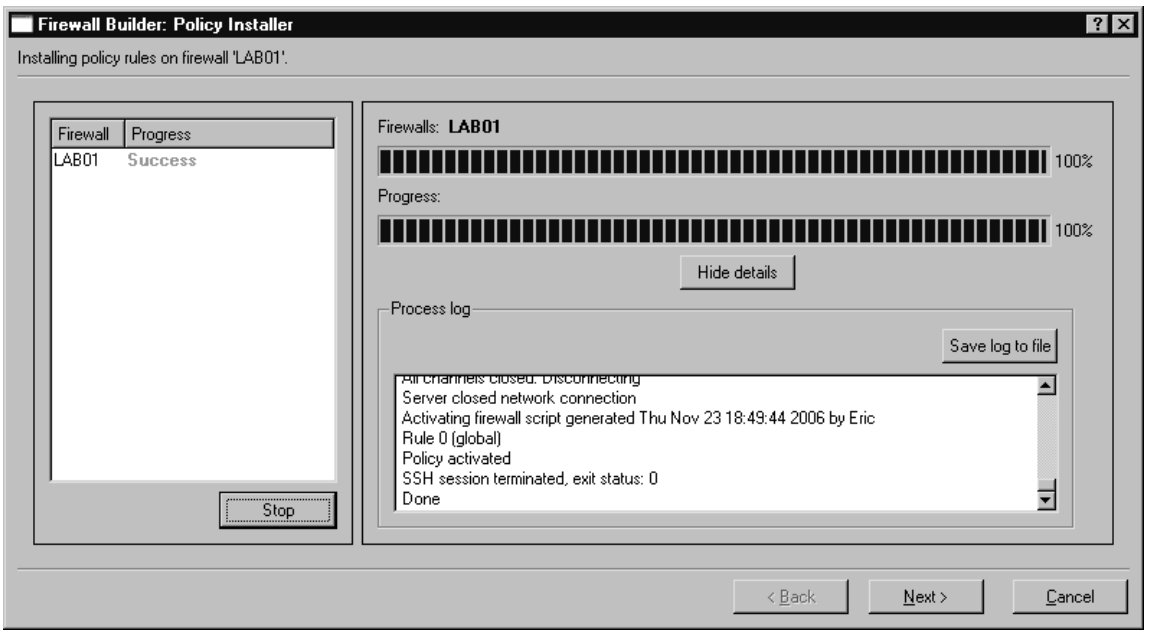
Anyone who has spent any time configuring firewalls has learned the hard way to be very careful when configuring the rules. It is always a good idea to create the rules to PERMIT administrative access before any others. This is because as soon as you configure the default policies to DROP, your SSH connection will no longer be permitted unless you have it added to the access list. If you forget to do this, you could find that you no longer have remote access to your firewall after applying the policy. If that happens, you won't even be able to remotely connect to update the policy and change the ACLs.

The next step is to tell FWBuilder where to find the SSH executables, because this is how FWBuilder uploads the configuration to the firewalls. You need to have SSH working on both the firewall and the FWBuilder console (assuming they are on different systems).

34. Select Edit | Preferences from the menu.
35. Select the **SSH** tab and click the **Browse** button.
36. Navigate to the location of your desired SSH utility (e.g., *plink.exe*) and click **Open**. Note that if you are using Windows for the FWBuilder host, you cannot select *PuTTY.exe*; you must use the command-line PuTTY program, *plink.exe*.
37. After selecting the SSH executable, click **OK**.
38. Right-click the **LAB01** firewall in the object tree, and select **Install**.
39. Select the Firewalls you wish to install to, and click **Next**.
40. Enter the username and password for the SSH connection.
41. All other fields are optional; however, it is recommended that you check "Store a copy of the fwb on the firewall." When satisfied with your choices, click **Ok**.

After the upload completes, you will get a status of "Success" (see Figure 9.24). Checking your firewall (*iptables –L*) shows you the new rules that are listed.

Figure 9.24 Policy Install Success



As you can probably see, once you have completed the up-front work of defining your objects, adding or modifying rules is simple. Additionally, unlike the other free GUI solutions, FWBuilder allows you to centrally and securely administer all of your (supported) firewalls from one location. When you use the aforementioned policy, Figure 9.25 shows a sample of the iptables rules that were generated.

Figure 9.25 FWBuilder Generated Chains

```
Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED
RULE_0     tcp  --  anywhere             192.168.1.12        tcp multiport dports https,http state NEW
RULE_1     tcp  --  192.168.1.0/24       anywhere            tcp multiport dports http,https state NEW

Chain RULE_0 (2 references)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere            LOG level info prefix `RULE 0 -- ACCEPT '
ACCEPT     all  --  anywhere             anywhere

Chain RULE_1 (3 references)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere            LOG level info prefix `RULE 1 -- ACCEPT '
ACCEPT     all  --  anywhere             anywhere
```

Notice that the default chains have rules matching the rule you configured in FWBuilder, with a target of *RULE_<RULE NUMBER>*. These additional chains are used to configure the logging. There is also a rule at the beginning of all chains to *ACCEPT* traffic related to an established session. This is generally desirable but is still configurable. To remote this automatically generated rule, select the firewall in the object tree and click on

Firewall Settings in the dialog area. There is a checkbox that is selected by default called "Accept ESTABLISHED and RELATED packets before the first rule." Although the FWBuilder policies you've configured can handle any basic rules you might need, there are still a few more bases to cover. If you need to NAT with your Linux firewall, configuring it with FWBuilder is easy. Follow these steps so that your Firewall will NAT all the traffic from the internal network to the DHCP address used on the outside interface. This configuration is also known as *source nat* because it is the source address that is being changed.

1. In the dialog area select the **NAT** tab.
2. Right-click and select **Insert Rule**. This will add a NAT rule number zero.
3. Drag your INTERNAL network object from the object tree to the **Original Src** column in the new NAT policy.
4. Drag the external interface on the firewall from the object tree to the "Translated Source" column in the NAT policy.

That's all there is to it. Save, compile, and install the new policy. Now traffic originating from the internal network will be NAT'ed to the IP on the external interface. Although this source NAT configuration will allow all your internal users to reach the internet, you will need to use *destination NAT* if Internet users need to reach an internal server. Because the internal server is using a private IP address (which is not routable on the Internet), you need to translate this destination to an IP address that the external users can reach. To configure packets destined for the firewall's single public IP address to an inside resource using destination NAT, follow these steps.

1. In the dialog select the NAT tab
2. Right click on the rule number zero of the existing NAT rule and select **Add Rule Below**.
3. Drag the firewall OUTSIDE interface into the Original Destination column of the new rule.
4. Drag the appropriate services (i.e. HTTP for web access) into the Original Service column of the new rule.
5. Drag the internal server into the translated destination column of the new rule.

Another nice feature is being able to create a time policy (e.g., if you only want the internal systems to be able to surf the Internet from noon to 1:00 P.M., you can easily make that adjustment.

1. In the object tree, right-click **Time**, and select **New Time Interval**.
2. In the "Name" field we'll call this rule **LUNCH**.

3. In the two time fields provided, enter a time for the rule to START and a time for the rule to STOP. In this case we will enter 12:00 and 13:00 and leave the date field as zeros. The day of the week can stay at -1, which means all days. When done, click **Apply**.
4. Drag the **LUNCH** time interval, form the object tree to the **Time** column of rule # 1.

Now, rule # 1 (which permits outbound Web surfing) will only be active from noon to 1:00 P.M. The ability to configure the rules to be active based on the time of day is a very powerful feature. If the organization is a strictly 8:00 A.M to 5:00 P.M type of place, you could configure the firewall to disable all access during non-business hours. Alternatively, certain non-business-related protocols (e.g., instant messenger, file sharing, and so on) could be enabled after the normal business day ends. While not the easiest of GUI to use, FWBuilder is definitely the most full featured, and the only one offering features you would expect to find in a commercial product.

Other GUIs

Although there are too many netfilter GUIs to cover them all extensively, I have tried to cover some of the best ones available. If none of the ones covered strike your fancy, or if you just like to experiment and see what else is out there, you might want to investigate some additional offerings. If you are running KDE look into Guarddog from www.simonzone.com/software/guarddog/#introduction, which is aimed at novice to intermediate users and offers the ability to define security policies based on logical groupings called network "zones." The Turtle Firewall Project **(**www.turtlefirewall.com/) allows you to administer your firewall host via a Web interface. While there is no substitute for a good understanding of the command-line configuration of iptables, for an uncomplicated firewall configuration many of these GUI's allow you to get your firewall up and running quickly and without having to read the iptables man page.

Smoothwall

Smoothwall (http://smoothwall.org/) is a firewall in it's own right, and is the site for SmoothWall Express. SmoothWall Express is a free, open source firewall solution. *Smoothwall.net* is the home of SmoothWall Limited, which produces several commercial security products including a version of the SmoothWall firewall. Smoothwall differs from the other solutions covered here, in that smoothwall is a dedicated firewall device. Other solutions using netfilter and optional GUIs to configure the firewall can be run on a workstation. You can still use the firewall system as a normal workstation, but it's not recommended. If you want to harden the firewall (as you should), you need to remove unneeded services and software from the system, and update all of the remaining software. Smoothwall takes a different approach in that all of this is done for you. When you install SmoothWall, it

wipes out the filesystem and installs a secured version of Linux on the hard disk, along with the SmoothWall software. The SmoothWall firewall has no GUI on the system, only command-line access and administration via the Web management interface. SmoothWall is meant to be a firewall and nothing more.

With that in mind, there are several advantages to this approach. Foremost, you don't have to learn how to harden your Linux distribution so that it will be secure enough to use. Further, unlike installing Linux and then learning iptables syntax, with SmoothWall you don't need to know Linux. The installation menu walks you through configuring the minimum settings so that you can then use the Web interface to configure the firewall functionality. You don't need to know everything about Linux to get SmoothWall up and running (though it never hurts). The fact that the SmoothWall firewall is already stripped down and unneeded software and services are removed means that you can get the maximum performance out of an old computer without having to spend a lot of time trying to tweak a full (normal) Linux distribution.

Installing Smoothwall

The simplest way to install SmoothWall Express is by downloading the *.iso* image from http://smoothwall.org/get/. It is advisable to read along with the manuals located at http://smoothwall.org/docs/. The documentation provided with SmoothWall Express is exceptional among free products, and all of the installation screens are shown in the PDF installation guide. This installation method is used as we walk though installing SmoothWall Express.

1. After burning this image to a CD-ROM, boot the prospective firewall with the CD-ROM in the drive.
2. The boot screen will look typical of many Linux distributions. It will warn you that installing SmoothWall Express will delete all data on the hard drive. To continue with the installation, press **ENTER**.
3. The installation then shifts into a DOS-like GUI interface. Navigation is accomplished using the TAB, arrow, and ENTER keys. You will be prompted to insert the installation CD and press **OK**. This is done in case your system cannot boot from a CD-ROM and you used a boot floppy to begin the installation. Either way, ensure that the CD-ROM is (still) in the drive, highlight **OK**, and press **ENTER**.
4. You have to select **OK** twice before the hard disk will be repartitioned and all data lost.
5. When prompted, select **Probe** to allow the installation routine to see what network cards it can detect.

SmoothWall uses a concept of interface colors to denote their trust level, and you will begin seeing them referred to in that fashion during the installation process. For example,

your inside interface is assumed to be the trusted traffic and is designated as the GREEN interface. Various dial-up or Indirect Defense Switched Networks (IDSNs) are designated as ORANGE interfaces. You can also have a combination of colors indicating your interface configuration. If you use an additional Ethernet interface as the untrusted (*OUTSIDE*) interface, in Smoothwall parlance that would be GREEN + RED.

After the files are installed successfully, you are given the opportunity to restore your configuration from floppy disks. This is useful if you are upgrading or migrating to new hardware. In this case, we select **OK**.

1. Select your keyboard layout (in my case "US," and select **OK**.
2. Select a hostname for the firewall (e.g., smoothwall) and select **OK**.
3. The next screen allows you to enter proxy server information in case you need to go through a proxy for the firewall to retrieve Web updates. If you are using a proxy, enter the appropriate information here; if not, select **OK**.
4. The next couple of screens allow you to enter configuration information for an ISDN or ADSL connection. The assumption of the installation process is that your *INSIDE* (trusted) interface will be an Ethernet interface, and the *OUTSIDE* (untrusted) interface will be either an Ethernet, ISDN, or ADSL. If you are using one of these, enter the appropriate information. If your *OUTSIDE* interface is a normal Ethernet interface (e.g., from a cable modem), select **DISABLE** for both the ISDN and Asymmetric Digital Subscriber Line (ADSL) configuration screens.
5. The next screen allows you to review and edit your network configuration. If you are using an Ethernet interface for both, you need to select **GREEN + RED** for the network configuration type. Check each menu option here and ensure that both interfaces have been recognized, have a driver installed, and have IP address settings. Commonly, the RED interface uses DHCP and the GREEN uses a static IP address, so that internal hosts can configure the firewall as their default gateway out to the Internet.
6. When you are satisfied with all the settings, select **DONE**.
7. You will be asked if you want to enable the SmoothWall firewall to serve as a DHCP server. This is the same configuration as most home firewalls, acting as firewall, gateway, and DHCP server. If you do not already have a DHCP server in your network, enable it. Fill in the desired values for the various fields. Most of the settings are not mission critical, but one setting to take note of is the lease duration. Too long of a lease duration and you will slowly loose IP addresses from systems that did not get the changes to release the address properly prior to going offline (such as from a crash or power outage). If the lease time is too long, this IP address attrition can exhaust the DHCP scope and leave no address available for other users. A 24-hour lease is not uncommon, and generally the larger the network the

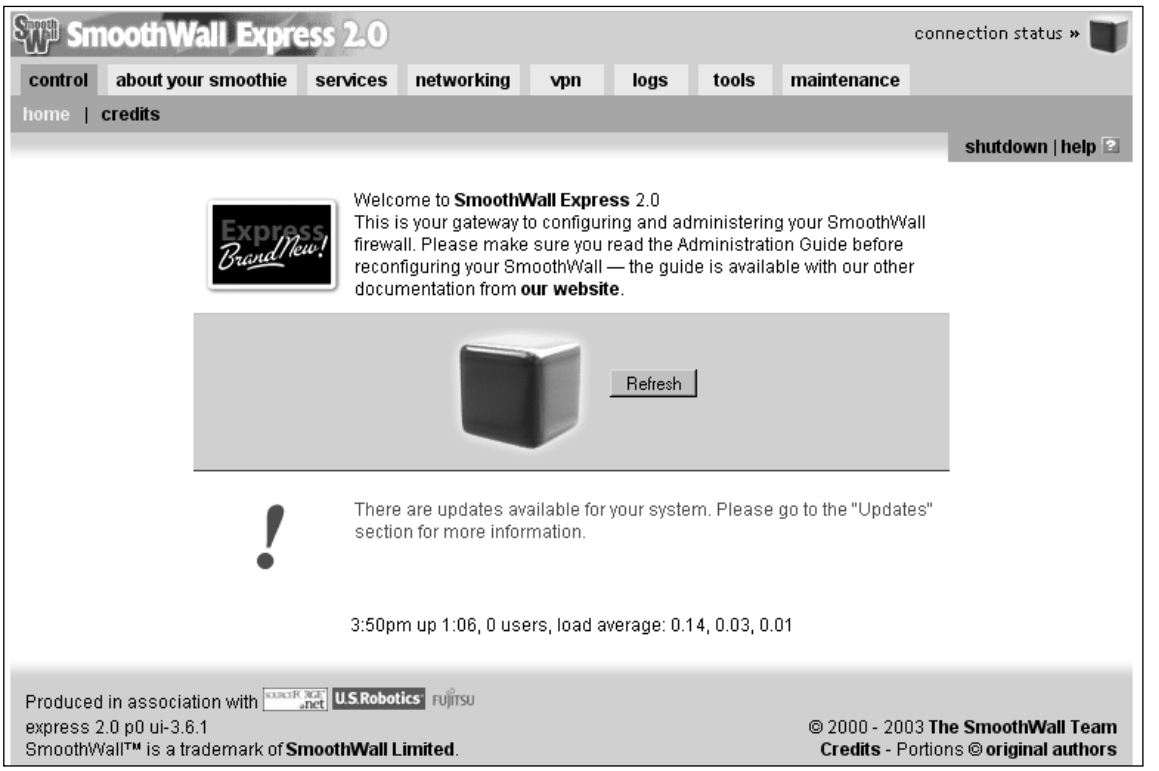
shorter the lease duration you will want. If you are unsure about DHCP, you can leave it disabled. The DHCP settings are easily configured later from the Web interface.

8. The next several screens allow you to enter the password for various accounts used by the firewall. Here are summaries of these accounts.
   - **Administrator** This is used for administering the firewall via the Web interface. This account is only for accessing the Web interface and cannot be used to login to the Linux OS on the firewall directly.
   - **root** This is a local Linux account that is used for command-line access on the firewall itself.
   - **setup** This is a local Linux account that is used to run the setup program, which automatically starts when you login as setup. The setup program allows you to configure some of the network settings if they need to be changed after the initial installation.
9. After you configure the final password, the CD-ROM will eject and the system will reboot. When the system comes back up, you can login directly via the console using the root account, or the preferred method is to login to the Web interface. The default Web interface is found at http://smoothwall:81 and the secure HTTP is found at https://smoothwall:441. Both the root and the setup account can also login via SSH, which is configured by default on port 222.

Configuring Smoothwall

When you first log into the smoothwall Web interface the screen will look like the one shown in Figure 9.26.

Figure 9.26 SmoothWall Web Interface



There is some information available before logging in, such as the number of users and average load on the firewall. As soon as you click on a menu item at the top you are prompted to authenticate with the Web admin user. By default, the account name is "admin." One of the first things you should do is enable SSH access, which is disabled by default. This allows you an additional way to manage the firewall if something goes wrong with the Web server or the firewall filters. You can enable SSH by clicking on the **Services** tab, and then selecting **Remote Access**. Next, place a check in the box next to SSH and click **Save**. You can verify what services are running by clicking the "About your smoothie" tab. There are three screens available under this tab. The *status* screen shows which services are running. The *advanced* screen shows more detailed information regarding memory usage, hard disk usage, network interface settings, and uptime. The *traffic graphs* screen shows input and output rates for all interfaces.

After enabling SSH, you should be able to connect on port 222. An example using *openssh* would be*:*

```
ssh <IP> -l root -p 222
```

Now that you have a backup way to get into the firewall, the next priority is to update the firewall. Although you don't have much of a configuration to warrant making a backup before applying the patches, it is still a good habit to get into. By selecting the **maintenance** tab and then the **backup** screen, you have a couple of options. The "Create backup floppy disk" button will write the configuration information directly to a floppy disk. Given the relatively unreliable nature of floppy disks, you should choose the "Create backup floppy image file" option. This creates and downloads an *.img* file to the system you are using for Web administration. You can store this file on a more reliable media, and then write the image to a physical floppy disk at a later date using a utility such as rawwrite. Once you have made a backup, you can safely apply the firewall updates.

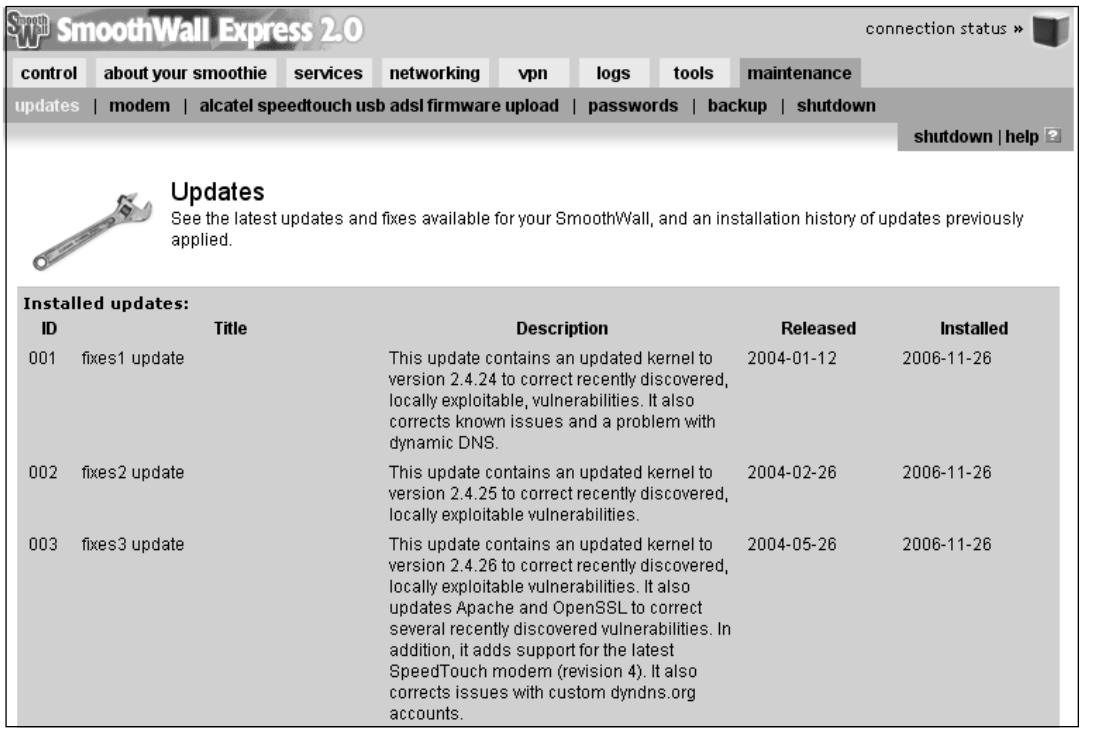
Firewall updates are another area where the SmoothWall people have made things as painless as possible. Click on the **maintenance** tab and you will see two sections on the **updates** screen. The top section shows *installed* updates and the bottom one shows *available* updates. To update the firewall, go to http://smoothwall.org/get/. In the "Latest Updates and Patches" section there is a small link called **updates archive**. Click that link and on the following page, download all the available updates to your local system.

NOTE

All of the updates must be applied sequentially; the SmoothWall updates are not cumulative updates. Apply update 1 first, then update 2, and so on, until you have applied all of the updates currently available for your firewall. At the time of this writing there were seven updates available for download.

The bottom of the **Maintenance | Updates** page has a box to upload an update. Click **Browse** and select the first update, and then click **upload**. The firewall automatically installs the patch as it is uploaded and, when finished, the page will refresh and show the updated listed in the "Installed updates" section. Continue this process until all available updates have been completed. A partial list of the successfully installed updates can be seen in Figure 9.27.

Figure 9.27 SmoothWall Installed Updates



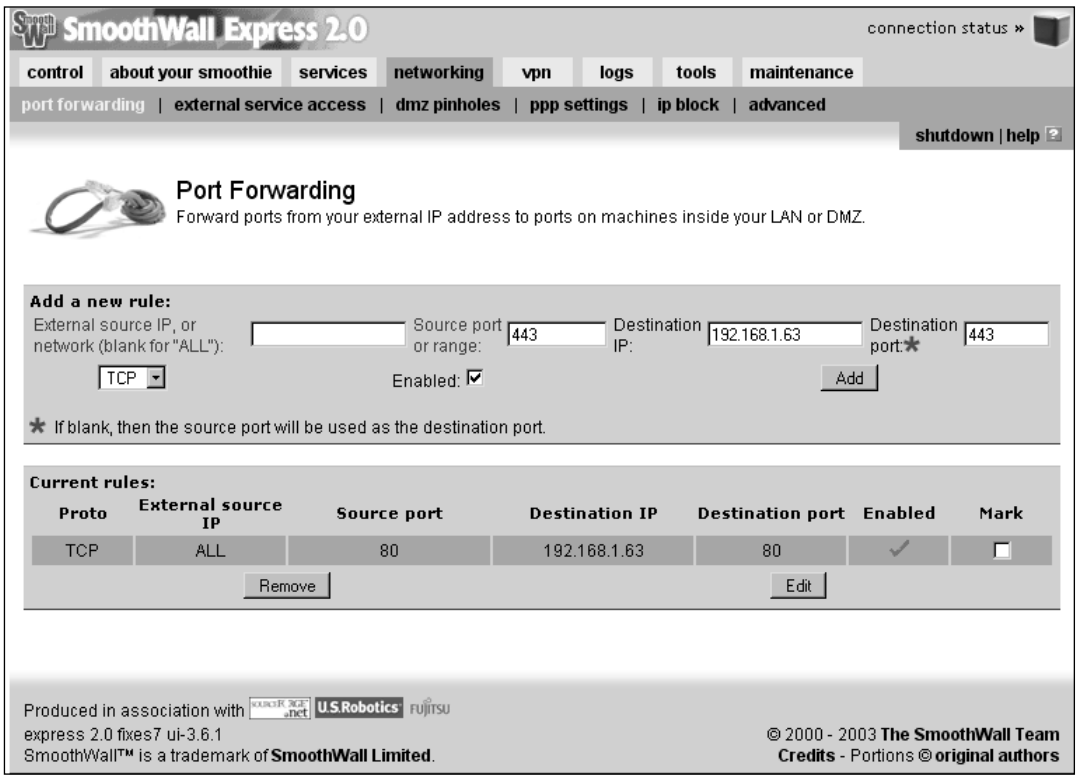
One final configuration option that should be a part of any firewall setup is providing the firewall with a good way to synchronize it's clock. Accurate time is important for many reasons, one of which is to make sure your logs have accurate time stamps. Without accurate time stamps, it will be more difficult, if not impossible, to reconstruct events later if there is an intrusion. You can configure the time source on the "Services" tab using the "time" screen. You should use the drop-down box to select the appropriate time zone. While SmoothWall does not give you the option to configure network time protocol (NTP) security, it does give you the option of using random public servers. In this fashion, even if it pulled time from one that was too far off (either accidentally or maliciously), the next time it is checked (a different server) would likely correct itself. To enable SmoothWall to retrieve the time from a public time server, check the **Enabled:** checkbox and then click **Save**.

With all of the basic administrative configuration out of the way, the actual rule configuration is next. SmoothWall relies heavily on the security level of the interfaces for access permissions. By default, all traffic will be blocked that enters on a RED interface and is destined for an address via a GREEN interface, unless it is specifically configured to be permitted. Similarly, traffic is blocked by default from an ORANGE interface to a GREEN interface. Traffic from a more secure interface to a less secure interface is permitted by default. This behavior is similar to several other commercial firewalls including the Cisco PIX/ASA. What

all this means is that for your users to access the Internet, you don't need to configure anything at all.

On the other hand, suppose you wanted to permit inbound access to a Web server (GREEN interface) with an IP address of 192.168.1.63 from any host on the Internet (RED interface). You would configure this by selecting the **Networking** tab and the **Port Forwarding** screen (see Figure 9.28).

Figure 9.28 SmoothWall Port Forwarding



Leave the source IP blank (for ALL) and enter a source port of 80. For the destination IP, enter the internal server's IP of 192.168.1.63, and for the destination port, enter 80 for HTTP. When finished, click **Add**. If you need to permit HTTPS, you need to repeat the process with 443 as the destination port. By using SSH to connect to the firewall directly (smoothware use port 222 for SSH), you can list the netfilter rules using *iptables –L* and see where the HTTP and HTTPS rules were added.

```
Chain portfwf (1 references)
target     prot opt source          destination
ACCEPT     tcp  --  anywhere        192.168.1.63      state NEW tcp dpt:http
ACCEPT     tcp  --  anywhere        192.168.1.63      state NEW tcp dpt:https
```

If you have three interfaces in a one-legged DMZ design, the DMZ interface is labeled as ORANGE. If you need to permit access from the DMZ into the trusted (GREEN) network, the process is a little different. You would then navigate to the **networking | DMZ pinholes** screen. The interface is very similar to the port forwarding with the exception that there is no field to specify the source port.

At times, an internal system's permissions may allow it to communicate with a device outside the firewall (on the RED interface); however, you may wish to block the communications completely. If you do this, any access by the blocked site will fail, even attempts to respond to an internal trusted system's request. You can configure this on the "Networking" tab, using the "ip block" screen. Enter the source IP address or name to block, and click **Add** to save the rule. You also have the option to enable logging for the blocked attempts.

With the basic firewall rules and maintenance configured, there are a few "extras" that are nice to see in a free product. One of these is the built-in Intrusion Detection System (IDS). Because it uses Linux as its base operating system, it conveniently includes Snort IDS; all you have to do is enable it. Enable Snort by selecting the "Services" tab, and then the "Intrusion Detection System" screen. Place a check next to Snort and click **Save**. The Snort alerts and other logs can be viewed on the "logs" tab. There are several subscreens that include a drop-down box to select what subset of logs you want to see, such as SSH, SmoothWall (which will show your recently applied patches), and several more. The "Web proxy" screen is only useful if you are using the Web proxy feature of the firewall. The "firewall" page shows all blocked connections and allows you to filter by month and day. Lastly, the IDS screen shows events logged by the Snort IDS. Unfortunately, SmoothWall Express does not support remote logging natively, while the commercial offering does. It does, however, allow you to export the log files to a text file. See Chapters 3 and 4 for more details on Snort.

Another nice option is the dynamic DNS support. There are various dynamic DNS services available that will allow you to use a consistent DNS name to refer to a system who's IP address is dynamic via DHCP. In order to do this, you typically must install a small program on the host system that will periodically contact the dynamic DNS server and alert them to your current IP address. The service then uses this information to update their DNS records so that people can locate the system via DNS. The SmoothWall firewall has the capability to perform these updates for you, to the major dynamic DNS providers. You can configure dynamic DNS support by selecting navigating to the **services | dynamic dns** page. Use the drop-down menu to select the dynamic DNS service you are using, fill in the rest of the information, and click **Add**. The firewall will then make the updates to the service and all of the hosts to IP mappings can be maintained in one place rather than having to install an agent on all of the systems that need dynamic DNS functionality.

SmoothWall Express is a very well put together firewall package. The documentation is very good, and the setup and management is straightforward and understandable with having to know anything about the underlying operating system. With all of the advanced features such as traffic graphs, intrusion detection, and respectable logging, it deserves a top spot on

the list of contenders for "best free firewalls." If you want the efficiency of running your firewall on Linux without having to learn how to secure your Linux installation, give SmoothWall a try.

Providing Secure Remote Access

Sooner or later odds are good that you will either want or need the ability to work remotely. Providing remote access must be undertaken very cautiously, because, as soon as you allow an employee to connect to the corporate network, you have to some degree, extended your network boundary to their workstation. This means your network's security is now only as good as the security of the remote user's system or network. In many cases this borders on no security at all. That is why remote access must only be granted after careful consideration and planning. While the different types of remote access pose different levels of security risk, there are some planning and configuration steps that are common to all of them.

The first task is to determine what type of remote access is appropriate. With a virtual tunnel network (VPN), it is as if the remote workstation is on the corporate network. This generally provides the greatest level of functionality, but also poses the greatest risk. If the remote system is compromised, an attacker is effectively inside your corporate network. While there are steps you can take to mitigate these risks, they may be time- and effort-intensive. To plan, configure, and properly secure a VPN solution is the most involved choice of the various remote access solutions you could provide.

Another option is to provide remote desktop functionality. This would allow a remote user to see and use the desktop of a system at work. A remote desktop acts as if the user is at work, while a VPN acts as if the user's computer is at work. This type of solution is slightly easier to implement, because you can typically isolate the traffic that needs to be permitted through the firewall to a single TCP port. Many of the same risks exist, however, in that if an attacker manages to gain access to an internal desktop remotely, it is usually easy for them to move information out of the network or otherwise cause mischief. Another key consideration with this type of solution is that you need to have a computer at home and a computer at work. With the VPN option, you only need to use one system, so if the user has a laptop, it can be used while they work remotely.

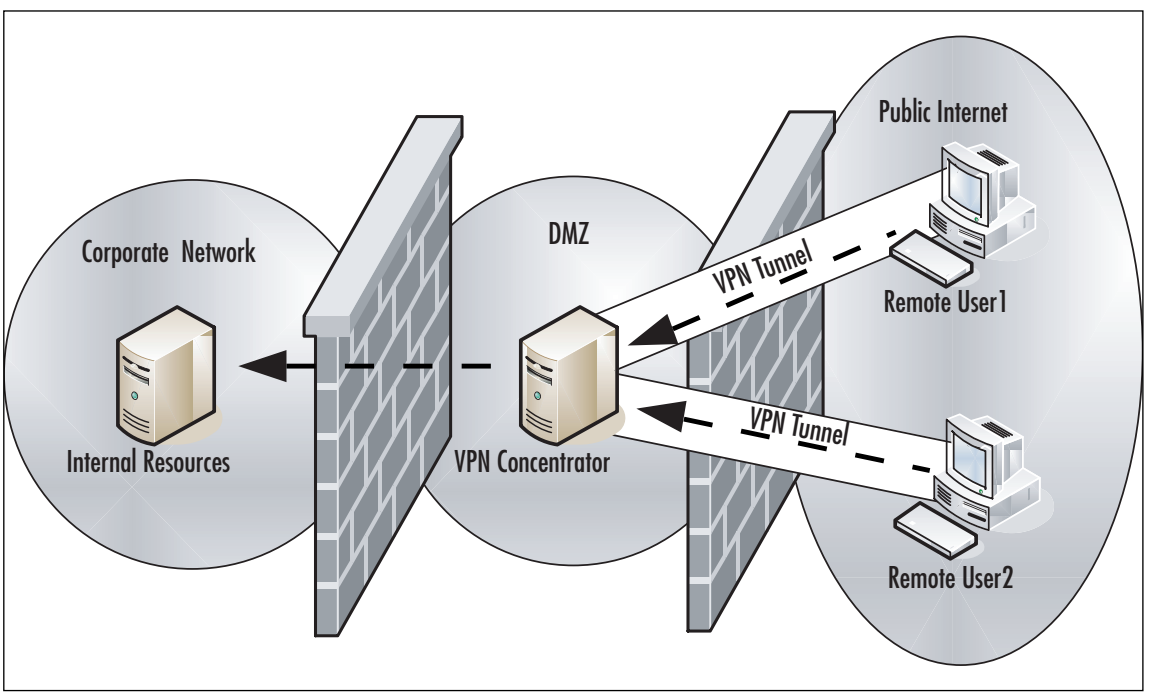
The last and least functional option is that of a remote shell. Because most average users don't operate extensively (if at all) in a console (i.e., text only) environment, this type of remote access is generally most viable for network administration personnel. While it may be impossible for accountants to operate their accounting program without a GUI, many network tasks and most firewall administration tasks can be performed with only terminal access. Because the widely used Telnet protocol sends all data unencrypted, any sensitive tasks should only be performed using a secured protocol such as secure shell (SSH), or Telnet over a Secure Internet Protocol (IPsec) tunnel.

Providing VPN Access

A virtual private network (VPN) is exactly what it sounds like, the network connection you create is virtual, because you can use it over an otherwise public network. Basically, you take two endpoints for the VPN tunnel, and all traffic between these two endpoints will be encrypted so that the data being transmitted is private and unreadable to the systems in between. Different VPN solutions use different protocols and encryption algorithms to accomplish this level of privacy. VPNs tend to be protocol independent, at least to some degree, in that the VPN configuration is not on a per-port basis. Rather, once you have established the VPN tunnel, all applicable traffic will be routed across the tunnel, effectively extending the boundaries of your internal network to include the remote host.

One of your first considerations when planning to implement a VPN solution is the network design. Because the VPN tunnel needs two endpoints, one will be the remote workstation. The other will be a specially configured device for that purpose. This is generally called a VPN concentrator, because it acts as a common endpoint for multiple VPN tunnels. The remote systems will effectively be using the concentrator as a gateway into the internal network, as such the placement of the concentrator is important. In a highly secured environment, the concentrator is placed in a DMZ sandwiched between two firewalls, one firewall facing the Internet, and the other facing internally (see Figure 9.29). While this type of arrangement is the most secure, it takes more hardware to implement.

Figure 9.29 VPN Concentrator Design



Another way to place the VPN concentrator inside a DMZ is to use an additional interface on the firewall as the DMZ in a "one-legged" configuration. This saves you having to implement an additional firewall, but still provides some isolation between the concentrator and the rest of the internal network. If an attacker compromised a remote host who was VPN'd into the concentrator or compromised the concentrator itself, they would still have a firewall between them and the internal network. The least preferable option it to place the concentrator inside the internal network. With this type of design, if the concentrator is compromised, the attacker would have full access to the internal network, with no firewalls to inhibit their activities. With any of these designs, you will have to permit the required ports through the firewall and forward them to your VPN concentrator.

Another consideration is the type of VPN protocol you want to use. IPSec is still the most widely deployed VPN technology for good reason. One is interoperability. As a widely used and tested standard, IPsec will work with virtually any modern firewall and operating system. The disadvantage of IPsec is that it can sometimes be difficult to configure properly, and there is zero margin for error on the configuration. Both ends have to use the same parameters for encryptions, hashing, and so forth, or the tunnel cannot be established. SSL is an increasingly popular choice for VPNs, largely because of it's simplicity to implement.

Once you have chosen a design and VPN technology, you need to consider the administrative ramifications of offering remote access. Some level of training will be required, at the very least so that they can sue the VPN software. You should educate the users on good security habits as well. A determination will also need to be made as to whether remote users are allowed to use their own personal computers, or if they must use a company-provided computer for remote access. The former option carries with it many risks. When a remote user connects their personal computer to the corporate network (via a VPN) they may have spyware, a virus, or any number or potentially damaging conditions present on their system. Due to the fact that you probably don't have any administrative access to their systems, you may have no way to secure the personal systems even if you wanted to. This is why most companies require that only corporate resources be allowed to connect to the company network. In the case of remote users, this typically means a company provided desktop, but I have also seen instances of older desktops being sent home for remote access.

A final consideration is one of hardware selection. Normal desktop productivity applications typically place very little strain on an even remotely modern processor. The same is not true when it comes to VPN connections. A single VPN connection requires little overhead and rarely impacts the remote user's system unless it is especially underpowered. For the VPN concentrator, however, it will handle the encryption and decryption of multiple connections, in addition to managing the volume of network data that will be accessed through it. For this reason, if you anticipate more than just a couple of VPN connections to be used simultaneously, you will want to test and evaluate your hardware needs.

OpenSSL VPN

One of the most commonly used open source SSL VPNs is Open VPN, which uses TAP and TUN virtual drivers. For Linux version 2.4.*x* or later, these driver are already bundled with the kernel. Open VPN tunnels traffic over the UDP port 5000. Open VPN can either use TUN driver to allow the IP traffic or TAP driver to pass the Ethernet traffic. Open VPN requires configuration to be set in the configuration files. Open VPN has two secure modes. The first is based on SSL/TLs security using public keys like RSA, and the second is based on using symmetric keys or pre-shared secrets. RSA certificates and the keys for the first mode can be generated by using the **openssl** command. Details about these certificates or the private keys are stored in our *.cnf files to establish VPN connection.

The .crt extension will denote the certificate file, and .key will be used to denote private keys. The SSL-VPN connection will be established between two entities, one of which will be a client, which can be your laptop, and the other will be a server running at your office or lab. Both these computers will have .conf files, which define the parameters required to establish SSL-VPN connection.

For the server side, let's call the file tls-srvr.conf, details of which are shown in Figure 9.30.

Figure 9.30 Configuration of the *.conf File on Server Side

```
# may be used to delimit comment. (Line 1)
dev tun
# 12.23.34.56 is the IP address of Server   (Line 2)
# 12.23.34.57 is the IP address of the Client  (Line 3)
 ifconfig 12.34.56 12.23.34.57
#scripts will establish routes when a VPN is alive (Line 4)
up ./srvr.up
# the script will be for server (Line 5)
tls-server
# Diffe –Hellman parameters (Line 6)
dh dh1234.pem
# Certificate Authority File (Line 7)
ca  my-ca.crt
 # Our certificate / Public Key (Line 8)
cert srvr.crt
# Private Key  key of Server (Line 9)
key srvr.key
# Open VPN uses port 5000 by default
# Each VPN must use a different Port (Line 10)
Port 5000
#UID and GID should be initialized to "nobody"  (Line 11)
user nobody
group nobody
#Verbosity Level  (Line 12)
# 0 --quite except for fatal error
#1 – mostly quite , but can display non-fatal network error
#3 – Medium output, good for normal operations
#9 – Verbose, good for trouble shooting
verb 3
```

The configuration of srvr.up, which is mentioned after line 4, is shown in Figure 9.31.

Figure 9.31 Configuration of the srvr.up File

```
# !/bin/bash
route add -net  12.0.1.0 netmask 255.255.255.0 gw $5
```

The *.cnf file (let's call it clt.cnf) on the client side will look similar to Figure 9.30. However, there will be modifications in some of the parameters in the file. After line 3, the parameters of ifconf will change to ifconfig 12.1.0.2 12.1.0.1 # from client side to server side. 12.23.34.57 is the IP address of the client, and # 12.23.34.56 is the IP address of the server.

After line 4, modification will be

```
up ./cnt.up
```

After line 5, modification will be

```
tls-client
```

Again, the certificate on the client side will point to the certificate of the client. If local.crt is storing the certificate of client and the private key of client is key local.key, then

```
cert home.crt
key local.key
```

will have to be added after line 8 and line 9.

The remaining part of the configuration file for the client side will remain the same.

The configuration of the clt.up to start a VPN server is shown in Figure 9.32.

Figure 9.32 Configuration of the clt.up File

```
#!/bin/bash
route add -net 12.23.34.0 netmask 255.255.255.0 gw $5
```

Once these files are configured, to start a VPN at the server side execute the command

```
$ open vpn -config tls-srvr.cnf
```

and similarly to start at the client side, use

```
$ openvpn -config tls-clt.cnf
```

Pros

SSL VPN is one way to transfer the information since a web browser can be used to establish an SSL VPN connection. Since SSL VPN is clientless, it will result in cost savings and can be configured to allow access from corporate laptops, home desktops, or any computer

in an Internet café. SSL VPNs also provide support for authentication methods and protocols, some of which include:

- Active Directory (AD)
- Lightweight Directory Access Protocol (LDAP)
- Windows NT LAN Manager (NTLM)
- Remote Authentication Dial-In User Service (RADIUS)
- RSA Security's RSA ACE/Server and RSA SecurID

Many SSL VPNs also provide support for single sign-on (SSO) capability. More sophisticated SSL VPN gateways provide additional network access through downloadable ActiveX components, Java applets, and installable Win32 applications. These add-ons help remote users access a wide range of applications, including:

- Citrix MetaFrame
- Microsoft Outlook
- NFS
- Remote Desktop
- Secure Shell (SSH)
- Telnet

However, note that not all SSL VPN products support all applications.

SSL VPN can also block traffic at the application level, blocking worms and viruses at the gateway. SSL VPN is again not bound to any IP address; hence, unlike IPsec VPN, connections can be maintained as the client moves. SSL VPN differs from IPsec VPN in that it provides fine-tuned access control. By using SSL VPN, each resource can be defined in a very granular manner, even as far as a URL. This feature of SSL VPN enables remote workers to access internal Web sites, applications, and file servers. This differs from IPsec VPN, since the entire corporate network can be defined in a single statement. SSL-based VPN uses Secure HTTP, TCP port 443. Many corporate network firewall policies allow outbound access for port 443 from any computer in the corporate network. In addition, since HTTPS traffic is encrypted, there will be limited restrictive firewall rules for SSL VPN.

Cons

As you know, SSL-based VPN offers a greater choice of client platforms and is easy to use. However, an organization that wants to be sure their communication channel is encrypted and well secured will never assume that any computer in an Internet café is trusted. This in turn requires a trust association with an un-trusted client connection. To address the concern

of an untrusted client, whenever a client from an untrusted platform connects to the VPN, a small java applet is downloaded to the client that searches for malicious files, processes, or ports. Based on the analysis of the computer, the applet can also restrict the types of client that can connect. This may sound feasible theoretically; practically, it requires the mapping of policies of one anti-virus and anti-spyware tool into an endpoint security tool used by VPN. In addition, these applets are prone to evasion and can be bypassed. However, note it carefully; you also need to have administrative access to perform many of the operations like deleting temporary files, deleting cookies, clearing cache, and so forth. If you have administrative rights in an Internet café, be assured that the system will be infected with keystroke loggers, sophisticated malicious remote access tools like Back Orifice using ICMP as a communication channel and RC4 to encrypt the payload.

By using SSL VPN, a user can download sensitive files or confidential, proprietary corporate data. This sensitive data has to be deleted from the local computer when an SSL VPN is terminated. To ensure the safety of confidential data, a sandbox is proposed and used. A sandbox is used to store any data downloaded from a corporate network via SSL VPN. After the SSL VPN session is terminated, the data in the sandbox is securely deleted. After a session is terminated, all logon credentials require deletion as well. You know that SSL VPN can be established even from a cyber café. It might happen that a user can leave the system unconnected. To prevent such issues, periodic authentication is required in some systems. As SSL VPN works on the boundary of Layers 4 and 5, each application has to support its use. In IPsec VPN, a large number of static IP address can be assigned to the remote client using RADIUS. This in turn provides the flexibility to filter and control the traffic based on source IP address. In the case of SSL VPN, the traffic is normally proxies from a single address, and all client sessions originate from this single IP. Thus, a network administrator is unable to allocate privileges using a source IP address. SSL-based VPN allows more firewall configurations as compared to IPsec VPN to control access to internal resources. Another cause of concern with SSL-based VPN is packet drop performance. IPsec will drop the malformed packet at the IP layer, whereas SSL will take it up the layer in the OSI model before dropping it. Hence, a packet will have to be processed more before it is dropped. This behavior of SSL-based VPN can be misused, used to execute DoS attacks, and if exploited, can result in a high capacity usage scenario.

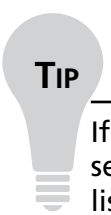
Using the X Window System

X window is the underlying management system for most Unix and Linux GUIs. It takes an entirely different architectural approach than a Microsoft Windows system, in that the X Window system is set up in a client-server architecture from the beginning, similar to VNC. When reading the X Window documentation, you will find that they user the terms server and client in the reverse of what would seem intuitive, meaning the server is where the display is being generated, not the remote machine you are connecting to. Most current implementations are based on the work of the *X.Org foundation* (http://x.org), which is the open

source implementation of the X11 protocol. A closely related project is the XFree86 Project (www.xfree86.org), which is the open source version of the *X Window system* (which uses the X11 protocol). X11 is the protocol that is used to transfer information about the GUI between the server and the client. The end result of these design decisions is that much like Windows' built-in terminal server support, two Linux systems can remotely access each other via a GUI virtual desktop.

You can configure the X Window System to permit connections from remote systems without any third-party software. While this works, the evolution of desktop Window Managers and common software packages has rendered this method inefficient. A much more robust way to accomplish the same thing is using NX technology developed by *NoMachine*, which is a highly optimized process and protocol to make X sessions available remotely. NX is available for free (client and server) from www.nomachine.com/download.php. Commercial variations are also available. You can see the differences between versions, and thus see what the limitations of the free version are at www.nomachine.com/features.php. The big limitation is that the *NX Free Edition* limits you to only two concurrent connections. In most cases this won't be much of a limitation unless you are trying to use it as a full-blown terminal server solution rather than just a remote access mechanism. An open source version of the NX server is called FreeNX and is available from http://freenx.berlios.de/. FreeNX does not support relaying sounds to the client (while the NoMachine server does). This is the recommended server and the one used in the following examples.

To set up the FreeNX server, download and install FreeNX using whatever method is appropriate for your Linux distribution. For Fedora Core 5, I used *yum install freenx*. Yum (the package installer for Fedora Core 5) will automatically check for and install any dependencies. The aforementioned command also installed the core *nx* package, and *expect* as dependencies. In the case of Fedora Core 5, there is no need for any further installation. Depending on the distribution you are using, the installation may be more involved. Most of the major distributions should have packages available that make the installation relatively painless. You will need to have SSHD listening on port 22 in order for NX to work properly.



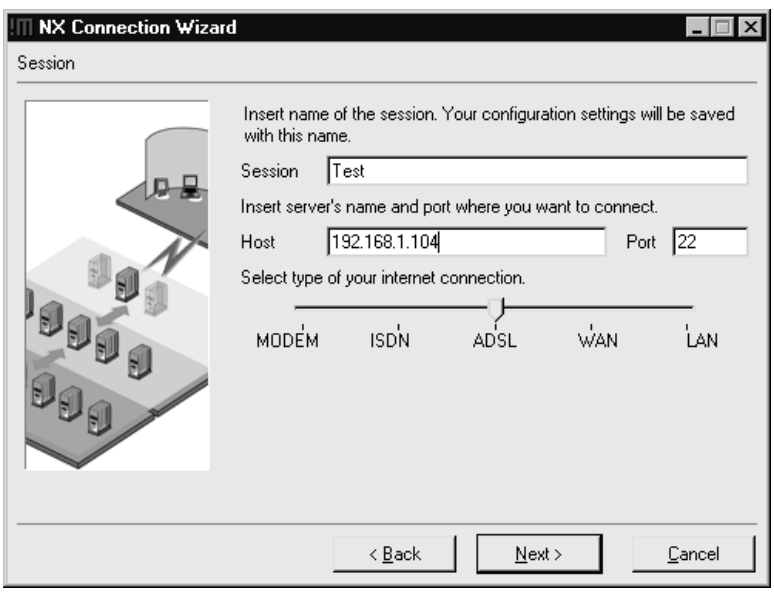
TIP

If you check the running services in an attempt to verify that the FreeNX server was installed successfully (via *chkconfig* or the like), you will not see it listed. This is because the server FreeNX server does not sit and listen on a port for an inbound connection like most services. Instead, when you login via SSH (this is why you need SSHD running on port 22), it logs you in as a special user (NX). That user's profile is configured such that it executes a process to start up the server and listen for the inbound connection.

With the server configured, the next step is to download and configure the client to make a connection. While there are alternate clients available, including some command-line clients, we will go with the original NoMachine client, which is installed on Windows XP for this exercise. Download the client and follow these steps to make a connection to the Linux FreeNX server:

1. Run the installation program. The installation is unremarkable, asking for all the standards prompts such as a license agreement, and choosing an installation directory, the option to create a desktop icon, and so on.
2. When you first run the NX Client for Windows shortcut, it will launch the NX Connection Wizard, because you have no sessions defined. The wizard will walk you through establishing a connection. On the first screen of the wizard click **Next**.
3. The next screen allows you to configure a name for the session. All of the connection settings will be saved under this name for future use. This window also asks for the host and port. Unless you have changed it, leave the port at the default of 22, and configure the appropriate host name or IP address in the host field. This screen is shown in Figure 9.33.

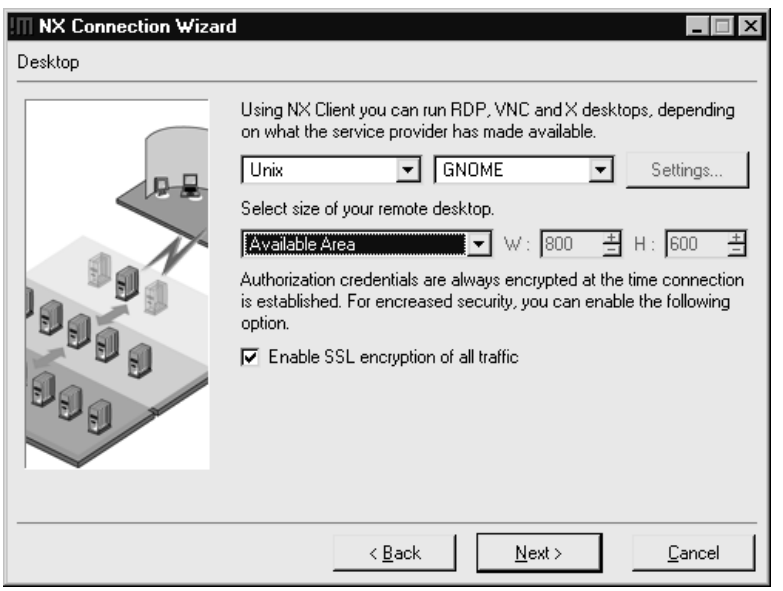
Figure 9.33 NX Client Wizard Session



4. After entering the required information, click **Next**.
5. The Desktop setting window is next. Select the OS you will be connecting to, along with the window manager. In this case it was **Unix** and **GNOME**. This

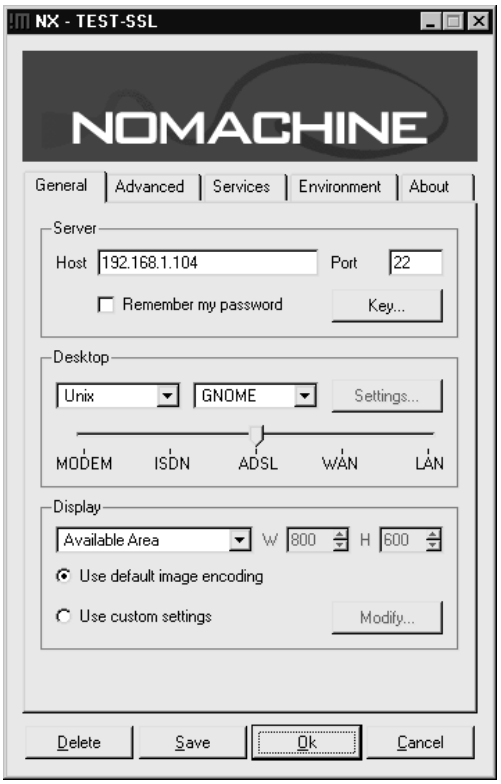
window also gives you the option of enabling SSL encryption of all traffic. Unless you have a reason not to have additional security, you should enable this option. If you do not enable encryption, all of the X11 data will be sent unencrypted, meaning that someone with a sniffer could capture and reconstruct everything sent between the client and the server. This screen is shown in Figure 9.34.

Figure 9.34 NX Client Wizard Desktop



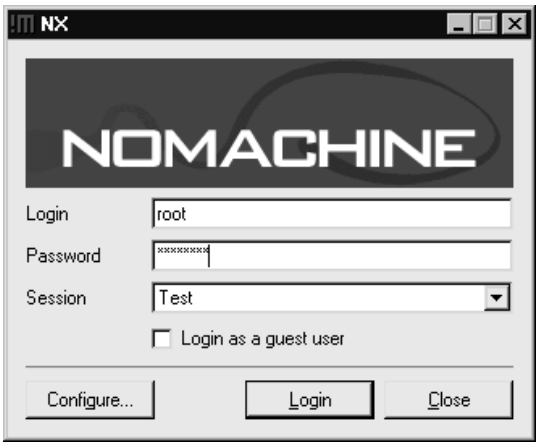
6. When you are satisfied with your settings, click **Next**.
7. The next screen of the wizard is the final one. The options are to **create shortcut on desktop**, and/or **show the advanced configuration dialog**. In order to enable SSL encryption for the connection, you will need to select the **show the advanced configuration dialog** checkbox and then click **Finish**.
8. The advanced configuration dialog is shown in Figure 9.35.

Figure 9.35 NX Client Advanced Dialog



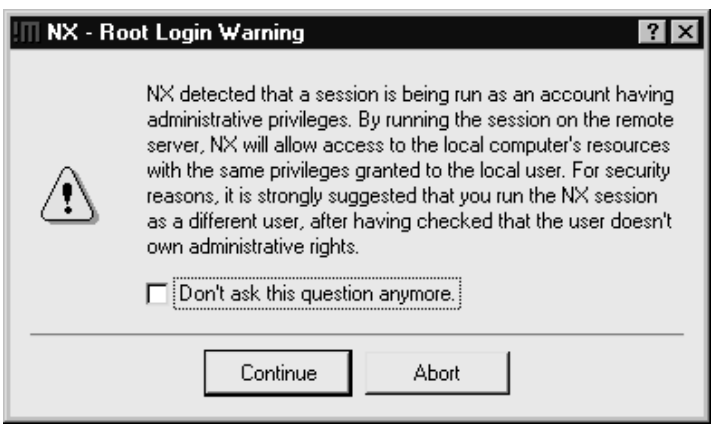
9. On the "General" tab in the "Server" section, click on **Key**. You must have a copy of the *client.id_dsa.key* file, which will be found on the server in either */etc/nxserver/* or */var/lib/nxserver/home/.ssh/*.
10. On the Key Management screen, click **Import**, and select the *client.id_dsa.key* file and then click **OK** followed by **Save**.
11. Finally, click **OK** to commit your settings for this session.
12. After completing the advanced configuration, you will see the client login window shown in Figure 9.36. Enter the appropriate login name and password, and click **Login**.

Figure 9.36 NC Client Login



If you login as a user with administrative privileges, you will be given a warning (see Figure 9.37) that this is not best practices and that it would be more secure to use a non-administrator account. If you get this warning, you can click **Continue** to proceed with the connection anyway.

Figure 9.37 NC Client Login Admin Warning



You will be prompted to verify the key fingerprint of the host you are connecting to, which can be done by clicking **Yes**. After a period of encryption and authentication negotiation, you should see the remote desktop. You will quickly notice that the desktop that is spawned is a "new" fresh desktop just for this login, not a view of an existing logged-in user's desktop. This default behavior is different than both Terminal Services and VNC, which both connect you to an existing session if one is present. While NX doesn't behave this way natively, there are some workarounds to try and modify this behavior. Hopefully,

future releases will simplify this option and make it easier to configure as the current workarounds are not elegant.

If you are in full screen mode such that you cannot see your real local desktop, there are some keyboard shortcuts available to accomplish some commonly needed tasks. These are outlined here.

- **CTR+ALT+F** Toggles full-screen mode.
- **CTRL+ALT+T** Shows the terminate/suspend dialog. If you terminate, the session is closed, while if you select suspend, you will be able to open the same session next time.
- **CTRL+ALT+M** Maximizes or minimizes the NX client window
- **CTRL+ALT+Mouse** Drags the desktop viewing area, so you can view different portions of the desktop. This is useful if the remote desktop has a higher resolution than your local resolution.
- **CTRL+ALT+Arrows CTRL+ALT+Keypad** – Will move the viewport by an incremental amount of pixels.
- **CTRL+ALT+S** It will activate "screen-scraping" mode, so all the GetImage originated by the clients will be forwarded to the real display. This will allow you to take a screenshot of the remote desktop, to your local clipboard. If you press the sequence again, nxagent will revert to the usual "fast" mode.
- **CTRL+ALT+E** Enables lazy image encoding for improved speed.

Summary

In this chapter, we examined a multitude of methods to secure your network perimeter and provide you, as the administrator, the access that is needed to administer the network. The Linux built-in firewall netfilter was covered extensively due to it's power and flexibility, not to mention availability, as a free stateful firewall. In addition to iptables, we looked at several GUI front ends that allow you to manage the netfilter firewall without knowing the iptables command line syntax. With your perimeter secured, the next step was to establish a secured doorway, so that you could sit at home and take care of the network. With command-line access via SSH, and Windows Terminal Services offering a remote desktop, FreeNX rounded out the offering by offering multiple remote desktop sessions from the same server.

Armed with this knowledge, there is no excuse to not have some type of firewall for protection on any and all unsecured connections. I say unsecured, not Internet intentionally, because any business partner, home user network, or the Internet are all considered untrusted, meaning you have no or incomplete administrative control over the security of the network you are connected to. Ultimately, you have no way to guarantee or enforce the proper security controls of an untrusted network. The sad fact is, if you have an Internet connection and don't have any type of firewall between a computer and the Internet, odds are very high that you have already been compromised. For other types of untrusted connections your odds may be better, but you're still gambling if you don't take steps to protect your network and systems.

Solutions Fast Track

Firewall Types

- ☑ In the networking sense, a firewall is basically any component (software or hardware) that restricts the flow of network traffic.
- ☑ Some firewalls are notoriously limited in capability, and others are extremely easy to use.
- ☑ To permit or deny traffic based on which network device is the sender or recipient and what ports are being used, you would use a packet-filtering firewall

Firewall Architectures

- ☑ The most securely configured firewall in existence will not provide much protection if the underlying network was not designed properly.

- ☑ A *screened subnet* is the simplest and most common firewall implementation. Most small businesses and homes use this type of firewall
- ☑ The one-legged demilitarized zone (DMZ) still has the advantage of cost, because you are building a DMZ using only a single firewall.
- ☑ The true DMZ is generally considered the most secure of firewall architectures. With this design, there is an external and internal firewall. Between the two is sandwiched any Internet accessible devices.

Implementing Firewalls

- ☑ netfilter is the built-in component that performs the firewall logic. iptables is the command-line interface used to configure the netfilter ACLs.
- ☑ Many GUI interfaces exist with widely varying degrees of functionality and complexity. My suggestion here is choose the simplest one that will do what you need it to do. In all likelihood the "right" one will change for you over time.
- ☑ SmoothWall sits in a class of its own, due to the fact that it turns a PC into a dedicated firewall appliance that is completely configurable without ever logging into the underlying Linux operating system.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How do I make the NX Client connect to an existing session?

A: Currently, the most oft used solution is to have the local user use an NX session (to localhost). That way, all of their activities are always within an NX session, making management (terminate, pause, resume) much simpler. Again, it's not elegant but it works adequately.

Q: I like the features of a particular flavor of VNC (or any remote access method) but it doesn't support encryption. What can I do to secure it?

A: One of the most common approaches is to use the SSH port-forwarding functionality. This is how many of the VNC variants are providing encryption behinds the scenes anyway.

Chapter 10

Linux Bastion Hosts

Solutions in this chapter:

- **System Installation**
- **Minimizing Services**
- **Additional Hardening Steps**
- **Controlling Access to Resources**
- **Auditing Access to Resources**
- **Sample Linux Host Configurations**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

The general concepts behind hardening a Linux bastion host are no different than they are for any other OS. You still need to minimize the installed software, update and patch the OS, and tighten the security settings. It is only the specific methods for accomplishing these tasks that vary with a Linux-based bastion host over a Windows-based one. The basic approach to hardening *any* bastion host (whether Linux, Unix, Windows, or other) can be summarized in the following *high-level* steps:

1. Planning comes before doing.
2. Remember the bastion host's role during all stages.
3. Leave only the minimum components on the system to get the job done.
4. Take what's left and make it as secure as possible, avoiding default settings where practical.

In some ways, securing Linux to serve as a bastion host can be harder than if you were using a Microsoft product, because there are so many options. If you aren't thoroughly comfortable with the landscape, the task of sorting out what tool and configuration to use can seem daunting. On the flip side, this same weakness—Linux's extreme flexibility—is also its main strength. Those same bewildering number of options provide you with near-infinite flexibility in how you accomplish your objectives. In this chapter we walk through the steps for hardening a Linux computer and preparing it to serve in the role of bastion host. We cover the specific steps you need to take and contrast those with the same steps you would use on a Windows system. It is assumed that the reader has read or is at least familiar with the chapter on Windows bastion hosts and that the reader has at least moderate understanding of and experience using Linux.

System Installation

With various flavors of Linux becoming so popular, deciding to use Linux as the OS for your bastion host has never been easier than it is today. The Internet is overflowing with helpful articles and Web sites to help guide you through setting up a Linux system. Many vendors are offering brand-new systems with Linux already installed. Several distributions of Linux are specifically configured to be user friendly and help a novice learn the ins and outs of Linux. We will go through the steps of hardening your Linux system on the assumption that you are installing it from scratch. If it is preinstalled on your system, you will need to evaluate whether starting over with a clean install with settings you choose is worth the effort. If you inherited the system, starting over might not be an option, but many of these procedures can still serve as good guidelines for how you can lock down your newly inherited system.

Disk Partitions

Linux has some specific requirements for disk partitions. All the applications that could be running at any given time can require a considerable amount of RAM to run. Typically, more memory is required than the machine has physically installed. Fortunately, operating systems have a technique to make it appear that more memory is present using *swap space*. This swap space is presented to the application as though it were more RAM. In Windows, the swap space is in the form of a swap file, which by default is located on the same partition as the system partition. Linux, on the other hand, expects the swap space to be on a separate physical partition. This swap partition actually gets formatted specifically as a Linux swap partition, not as a standard file system. Some Linux distributions allow you to use a normal file for swap space, like Windows, but using a separate swap partition provides some significant performance increases and increased stability.

The amount of swap space that is needed will vary based on the programs that are loaded and the functionality you are using in them. When you use a swap file, this means the size of the file will need to change regularly, causing increased reads and writes to the disk, above and beyond what is needed to manage the program data, simply to manage the swap file itself. By contrast, if you have a separate swap partition, there is no disk activity to manage the swap file, only the activity required by the programs using the swap space. Stability can be enhanced using a separate swap partition as well. If the partition containing your swap file fills up with data, it can prevent the swap file from growing and providing the needed swap space for a newly loaded program. A dedicated partition for the swap space provides the desired isolation from user data and ensures that the swap space will be available.

Choosing a Linux Version

Because there are so many different versions of Linux available, you have a lot of choices here. The different versions may have some of the same core components, but each offers different sets of software packages, often with an emphasis on a particular type of functionality, such as security testing, general office productivity, or software development platforms. These variations are called *distributions,* or sometimes even *distros* for short. A good source of information on the various distributions is www.distrowatch.com. This site includes a brief summary of what the distribution is trying to accomplish and includes links to the homepages and download locations. You can, of course, always do a Google search to find a distribution that does exactly what you are looking for. For a locally installed full installation, Fedora core (which we examine in the examples in this chapter) provides a lot of features and has extensive support documentation. If you just want to dabble, a live CD, such as SLAX, is a very user-friendly way to get your feet wet with Linux.

Choosing Distribution Media

You can find distributions that are capable of running off a CD-ROM, full installs to the systems hard disk, and distributions that can run off a USB drive or even a floppy disk. Each media type offers some security pros and cons, and not every distribution is available on every media type. If you need the features of a specific distribution that doesn't come on the media you prefer, you might need to make a compromise. The decisions of media type and distribution selection will be made concurrently, unless you have some very specific requirements. You will need to research the media options and choose one that fits your environment; we review some of the pros and cons of each type in the following sections.

Full Installs

With the traditional install to the systems hard disk, much like Windows, you will boot up an installation CD and walk through a guided install process. Although it wasn't always the case, most of the distributions that are intended to be installed to the hard disk offer GUI install programs that are pretty easy to get through. There is no great advantage to using this type of distribution other than the size of the hard disk will allow you to install a lot of extra software. In a DMZ on a bastion host, this isn't necessarily desirable, and in many cases one of the smaller, more streamlined distributions would be more appropriate.

On the down side, this type of install will have all the same disadvantages of a Windows bastion host—namely, that the entire system is sitting on the hard drive, and if an attacker manages to compromise the root account, she will be able to install a virus or Trojan on the system that can survive future reboots, in addition to tampering with the contents of what is running in memory. Typically, if you discover that your bastion host has been compromised, unless you have some means of ensuring that the contents of the hard disk have not been tampered with, you must reinstall the entire system from scratch to guarantee the system's integrity. This type of install isn't really any better or worse than if you were using Windows for your bastion host operating system.

CD-ROMs

There are many versions of Linux designed specifically to run from a CD-ROM, allowing you to turn virtually any machine into a firewall, router, or general-purpose PC. There is an obvious security advantage to having all your configuration information on read-only media. Even if a hacker manages to compromise the system, all it takes is a reboot and your configuration can be restored to its previous condition. The system can still fall victim to a virus or Trojan but only until it is rebooted. Further, if there is a hardware failure, restoring the system is only as difficult as moving the CD to a new system and rebooting.

This same advantage also serves as a disadvantage. If you burn the entire OS and configuration settings to a CD, any time you need to make any adjustment you would need to burn and load a new CD-ROM disk. The cost of the CD media probably isn't an issue,

because CD-ROMs are inexpensive, but such a configuration could hinder your ability to remotely administer the system. You would only be able to remotely make changes to the running configuration, whereas changes that will remain after a reboot will require someone local to burn and swap the CDs. If you needed to implement and test changes that required a reboot before they will take effect, this type of setup would make things a little more difficult. Finally, due to simple space limitations on a CD-ROM, you might not be able to fit all the needed software or functionality on a CD. All that being said, if the role of your bastion host allows the configuration and data to be relatively static, a live CD could be a very attractive option.

USB Drives

A Linux bastion host booting from a USB disk could offer the best compromise in security and flexibility. If you purchase a USB disk that includes a physical write-protect switch, you can make changes on the fly, as with a live system, and then write-protect the disk against modification when you are done. As the storage capacity of USB drives increases, you will be able to use a USB-based distribution that includes increasingly greater functionality. Whether you get the micro hard drive version or the solid-state USB disk, either will be more reliable than a diskette while offering the same benefit of allowing you to toggle the write-protect feature.

Diskettes

Although they probably won't be able to provide the functionality you are looking for in a Linux bastion host, many versions of Linux can fit on a 3.5-inch diskette. The primary advantage of these distributions is their very low resource requirements. Often these systems require only 8MB or 16MB of memory to function. The ability to toggle the write-protect switch on the diskette can also provide a high degree of configuration flexibility and security. Considering the unreliable nature of diskettes, they probably wouldn't be appropriate for any critical roles, though. Another disadvantage to diskettes is simply functionality. Whereas they can act as a simple router or firewall, they typically lack much in the way of features. Generally, these diskette-based distributions are single-purpose devices. Due to the space restrictions and small footprint, diskette-based distributions are almost always command line only, with no GUI for configuration or management.

Choosing a Specific Distribution

In conjunction with choosing the media to be used, you will need to choose the exact Linux distribution you want to employ. This decision can include a host of factors such as included features, personal preference, availability of documentation and support, and availability of technical support. Some distributions are known for being more beginner friendly or for supporting better security controls. One of the key differentiators for selecting a

Linux distribution as a bastion host is technical support. Some distributions, such as Red Hat Enterprise Linux, are available with full technical support, which can be a major consideration for a corporate bastion host.

NOTE

Red Hat is one of the oldest Linux distributions that is still active. The naming of Red Hat Enterprise Linux servers can be somewhat confusing. Red Hat Enterprise Linux (RHEL) is the commercial product that includes various levels of technical support. It comes in a few flavors, such as RHEL WS (appropriate in a workstation role), RHEL AS (which contains advanced server functionality such as database support or as an application server) and RHEL ES (appropriate in a small to medium-sized server role). See www.redhat.com/rhel/compare/ for more information. To further muddy the waters, the Fedora Core distribution is basically the same software as RHEL but without any support. Fedora Core is the freely downloadable consumer version. See http://fedora.redhat.com/ for information and downloads.

Once you have selected your preferred distribution, you should go through the installation process. If you are presented with any choices to install additional software, select No. We want to keep things to a minimum because our next steps will be removing any unnecessary software. The following steps assume you have the base Linux operating system installed and functional. I used the latest Fedora Core distribution for screenshots (V5) with the default GNOME window manager, but the procedures should be similar for most distributions.

Removing Optional Components

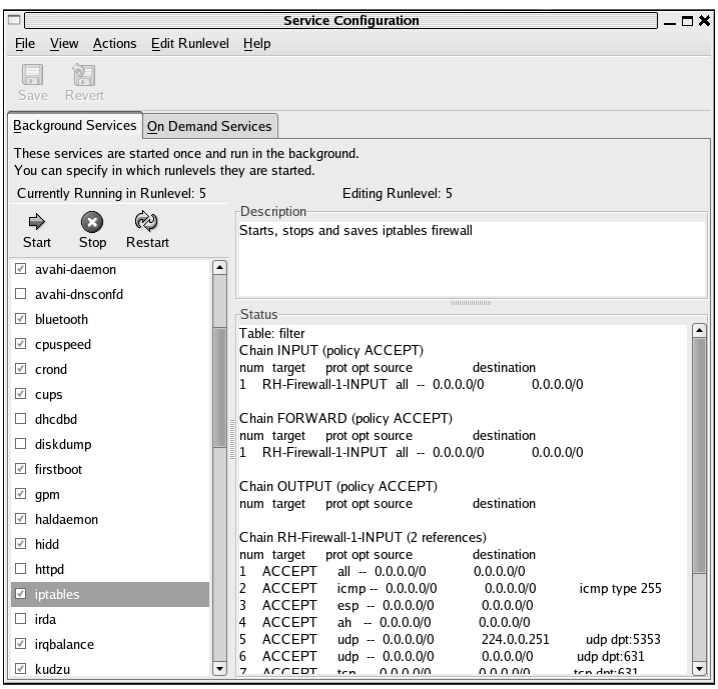
Windows has been the target of much scrutiny and ridicule for the size of its installation. Typically, Linux is seen as sleek and efficient. Well, this is partially true. The fact is that out of the box, if you don't take care to remove components, Linux can be pretty bloated. The default install of Fedora with none of the high-level optional software categories selected was just shy of 2GB, which isn't all that slim. The key difference between Linux and Windows is that with Linux you *can* remove or disable unneeded components (even including parts of the kernel itself) and strip the system down to a very lean and efficient machine. By contrast, tampering with the core OS files in Windows isn't really an option for most people.

Minimizing Services

Like a Windows system, your chosen Linux distribution will likely be installed with a large number of programs configured to run automatically, without human intervention. On Windows systems these programs are called *services;* on Linux they are called *daemons*. In an effort to minimize the amount of software that is running and thus reduce the number of software targets a hacker might try to exploit, you should disable any daemons that are not needed. Determining which daemons are needed will require some investigation and testing. You'll find a list and brief description of daemons in Fedora Core 5 at www.mjmwired.net/resources/mjm-services-fc5.html.

To configure the daemons, navigate to **System | Administration | Services**. (Fedora actually uses the Windows terminology to refer to the daemons as *services*.) This will open the GUI interface shown in Figure 10.1.

Figure 10.1 Daemon Control Panel



Highlighting a service in the leftmost pane will pull up a description and some additional details in the Description and Status panes, respectively. Disabling a service is as easy as removing the check mark next to the service in question, then clicking **Save** in the upper-right corner of the window and rebooting.

NOTE

Different services are handled differently with respect to the time at which your changes take effect. When changes are made to services managed through *xinetd*, *xinetd* is immediately restarted; thus your changes will take effect immediately. When the service is *not* managed by *xinetd*, the same is not true. In those cases you will need to either stop the service manually, go to a command prompt, type **telinit <*runlevel*>** and press **Enter** to reinitialize the run level, or simply reboot.

If you are administering the bastion host without access to a GUI front end, you will need to control daemons via the startup scripts. It's also a good idea to know how to control the daemons without the GUI because the GUI interface will vary from system to system. The methods for startup scripts can also vary from one distribution to another. The startup services in Fedora Core are managed by startup scripts located in /etc/rc.d/init.d. All you need to do is comment out the line that calls the specific script by inserting a # at the front of the line that calls it. You can also edit the individual scripts themselves for control of the way the services are initialized. The preferred way on Fedora Core is to use the *chkconfig* utility, which can be found in many distributions. You must be logged in as root to use this utility. Abbreviated output from *chkconfig —list* is shown in Figure 14.2.

Figure 14.2 Listing Services with *chkconfig*

```
[root@localhost ~]# chkconfig --list
NetworkManager  0:off   1:off   2:off   3:off   4:off   5:off   6:off
acpid           0:off   1:off   2:off   3:on    4:on    5:on    6:off
anacron         0:off   1:off   2:on    3:on    4:on    5:on    6:off
apmd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
avahi-daemon    0:off   1:off   2:off   3:on    4:on    5:on    6:off
avahi-dnsconfd  0:off   1:off   2:off   3:off   4:off   5:off   6:off
bluetooth       0:off   1:off   2:on    3:on    4:on    5:on    6:off
cpuspeed        0:off   1:on    2:on    3:on    4:on    5:on    6:off
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

As you can see, *apmd* (which is used for monitoring and logging the battery status) is configured to start for run levels two through five. Since our bastion host has no battery to monitor, we can *disable apmd* for all run levels by entering:

```
chkconfig --level 123456 apmd off
```

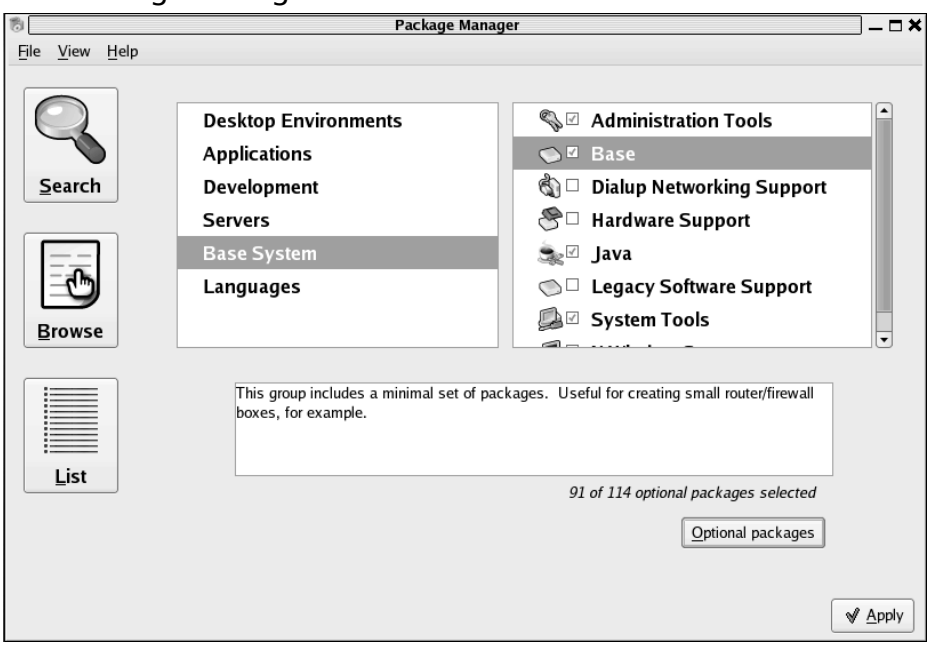
If you want to *remove* the service entirely form the startup script, type:

```
chkconfig -del apmd
```

Removing Optional Software

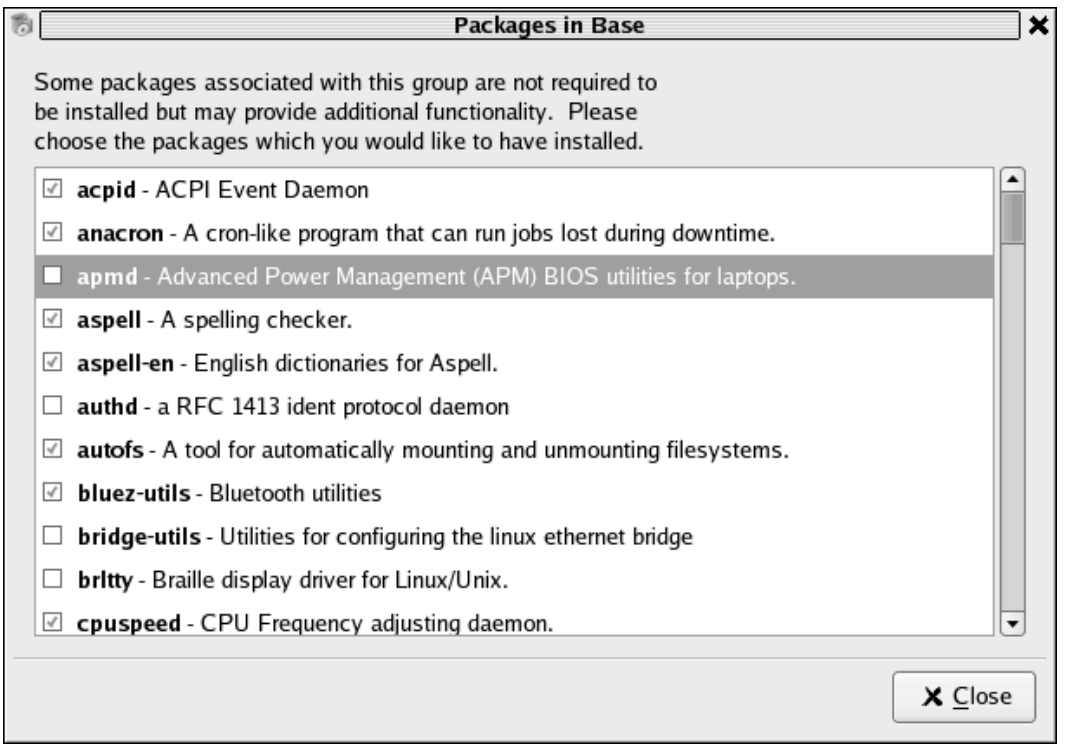
As with Windows, you will also likely end up with some software you don't need. Removing the software is most easily accomplished with Pirut, the built-in GUI tool you can access by navigating to **Applications | Add/Remove Software**. Once it finishes checking what is currently installed on the system, you see the Package Manager window, which allows you to search for packages, install packages, or remove them. When you use the **Browse** button, your interface is hierarchical and looks almost exactly like the one you used to install the operating system. The Package Manager window is shown in Figure 10.3.

Figure 10.3 Package Manager



To remove the aforementioned *apmd* package, highlight the **Base System** category in the left pane, and then highlight the **Base** group in the right pane. Click **Optional packages** and a new window will open showing all packages in the **Base** group, as shown in Figure 10.4. Simply clear the check mark next to **apmd** and click **Close**. Back at the **Package Manager** window, click **Apply**. Click **Continue** on the next window to verify your changes, and finally, click **OK** when the update is completed.

Figure 10.4 Package Listing



Packages can also be managed from the command line using the RPM Package Manager (RPM). You can view a list of all installed packages by entering:

```
rpm -q -a
```

For example, to install the *apmd* package, you would need to first obtain the package file itself (from www.redhat.com/download/mirror.html, for example) or use the RPMs that were included on the installation CDs. Then enter the following command to install *apmd*:

```
rpm -i apmd-3.2.2-3.2.i386.rpm
```

If the installation is successful, you should see output similar to the following;

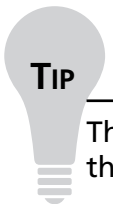
```
Preparing...                ########################################### [100%]
   1:apmd-3.2.2-3.2         ########################################### [100%]
```

You can even install a package directly from the Internet by specifying the full FTP or HTTP path as the path to the RPM, as follows:

```
rpm -i ftp://somesite.com/5/i386/RPMS/apmd-3.2.2-3.2.i386.rpm
```

To uninstall the package, you must use the package name, which can differ from the name of the RPM file. To uninstall *apmd*, enter the following command, using the *–e* switch, for *erase*:

```
rpm -e apmd-3.2.2-3.2
```



TIP

There are various tools for package management. Here's a brief summary of the tools included in Fedora Core 5:

pup GUI tool for updating software, accessed at **Applications | System Tools | Software Updater**.

pirut GUI tool for managing software packages, accessed at **Applications | Add/Remove Software**.

rpm Command-line tool for managing software packages.

yum Yellowdog updater modified; command-line tool for managing software packages.

yum Extender A GUI interface for YUM (install with yum install yumex).

Pirut and yum will automatically ensure you have the most current version of a package. Both of these will also automatically check and install any dependencies for the software you install. Rpm does not include this functionality you will need to check for dependencies manually when using rpm. YUM only works on RPM based systems, and not all systems will have YUM available/installed, therefore it is suggested that you understand how to manage packages with RPM even if you choose to use YUM for you day to day management.

There is no universal list to tell you which packages you should leave installed and which ones you should remove. You will need to evaluate each service based on your requirements. However, at a minimum the following services are ones you probably *do* need to have installed/running, unless you are very sure you don't need them:

- **haldaemon** Used for gathering and maintaining information concerning hardware devices
- **iptables** Manages IPTables firewalls
- **messagebus** Used for sending notification for certain system events
- **network** Manages the activation of network interface at boot-up
- **NTPd** Used to synchronize time via the NTP protocol

- **sshd** Runs OpenSSH server
- **syslogd** Used for system logging
- **xinetd** Manages the startup of services

Tools & Traps

Services to Be Avoided at All Costs

There are some services that you would never want exposed to an untrusted network such as the Internet. In most cases this means they should not be running on a bastion host. These services include the following:

- **portmap** Used to manage RPC connections
- **Telnet** Used for unencrypted remote console access
- **rsh, rlogin, rexec** Used for unencrypted remote console access
- **nfs, lockd, mountd, statd** Used for Network File System (NFS) and related services
- **lpd** Printer service

All these services are inherently insecure. A variety of alternatives is available that take advantage of encryption and superior authentication methods. If you absolutely must use any of these services, you should protect them by tunneling them in IPSec or equivalent.

Choosing a Window Manager

While we're on the subject of cleaning out unneeded software, it seems like a good time to discuss window managers. Because Linux is natively a command-line based system, the graphical desktop environment is not an integral part of the operating system. Unlike with modern versions of Windows, you can simply remove the GUI altogether. This has several advantages, the most noticeable of which is that you will free up system resources. The simplest and most easily implemented remote access solutions are also command line only, which further lends itself to doing away with the GUI altogether. Finally, by not offering up an X Window session for remote use, that is one less listening port that needs to be opened on your bastion host, which means increased security.

If you decide a graphical interface is something you must have, you still must choose one carefully. They are not all created equally. Generally speaking, the more full-featured and

visually appealing desktops are also the largest in terms of disk space and system resource requirements. Gnome and KDE are by far the most feature-rich, widely supported desktop environments. Gnome is not itself a window manager and comes standard with the Sawfish window manager. These two products also happen to be some of the largest and resource-intensive as well. If performance is a consideration, you will want to choose a lean and fast window manager such as IceWM (which is Gnome-aware) or Fluxbox. Some window managers can cause the loss of some features with different underlying desktops, so you will need to do a little research to make an informed choice.

Additional Steps

At this point, the basic system is in place. You have the desired software installed and should have the undesirable software removed. This covers only the most basic of hardening steps. The next steps will allow you to drill down into some of the more granular aspects of bastion host hardening. Now let's look at steps to synchronize time on the bastion host, keep the system patched, and more.

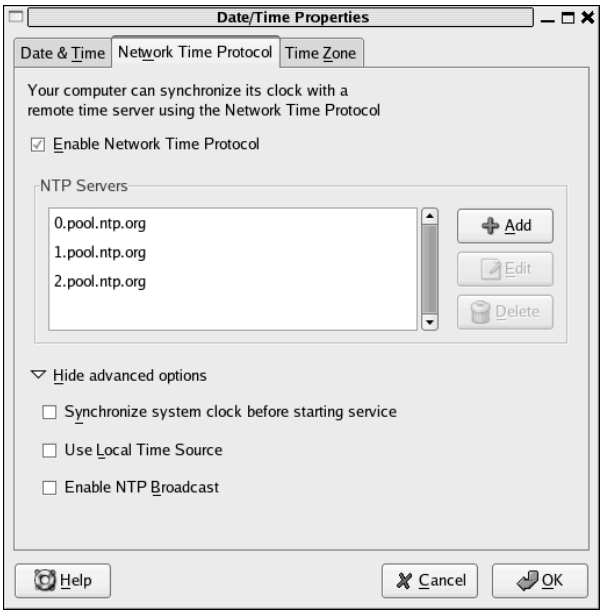
Configure Automatic Time Synchronization

Having an accurate system time is important to proper functioning of your bastion host. Accurate time on the bastion host is critical to having accurate audit logs, the ability to perform accurate forensics, and for maintaining secure (encrypted) communications. Fortunately, most distributions of Linux should support the newest versions of NTP by default. You will recall from the Windows chapter that different versions of NTP provide different levels of security:

- **NTPv1** No security features
- **NTPv2** Restrictions based on IP address, NTP traffic is unencrypted
- **NTPv3** Symmetric key encryption and authentication
- **NTPv4** Both symmetric encryption and PKI encryption

In the case of Linux, all you need to do is install the NTPD service, if it's not already installed, and then configure it to start automatically for the appropriate run levels. You can configure some of the NTP settings by navigating to **System | Administration | Date & Time**. You can then click the **Network Time Protocol** tab, as shown in Figure 10.5.

Figure 10.5 NTP GUI Configuration



Unfortunately, the GUI interface only lets your configure basic options, like adding time servers. You should make sure **Synchronize System clock before starting service** is the only option checked under **Show Advanced Options**. To configure the NTP security settings, you need to edit the file /etc/ntpd.conf and some additional files located in /etc/ntp/. For secure time synchronization with an NTP server, perform the following steps:

1. Edit **/etc/ntp.conf** and add the time servers you want to get time from; include the key to be used with each server, such as **server ourtimeserver.com key 12345**.
2. Ensure that **restrict default ignore** is in /etc/ntp.conf to set the default restrictions.
3. If needed, add a restrict line for any servers you want to *accept* NTP *requests* from.
4. Edit **/etc/ntp/keys** and configure a unique key to be used for each time server, using the desired algorithm; for example, **12345 A secretkey** would configure key #12345 as an ASCII string between one and eight characters long, with a value of *secretkey*. This key must be in the keys file of both the NTP server and the client for this to work.
5. Add the keys to the **/etc/ntp.conf** file. For example, to add key 12345 as a trusted key, enter **trustedkey 12345**.
6. Restrict the NTP configuration files to be owned and readable only by root.

You can view a detailed NTPD status by typing **ntpq –p**. If you enter **ntpq** and query associations, it will display the status of all associations, including an auth column indicating whether authentication is working properly, as shown in Figure 14.6. You'll find more information at www.ntp.org.

Figure 14.6 Viewing NTP Associations

```
# ntpq
ntpq> associations
ind assID status conf reach auth condition last_event cnt
===========================================================
  1 38100  9414  yes   yes   ok   sys.peer  reachable 1
  2 38101  9614  yes   yes  none  sys.peer  reachable 1
```

Patching and Updates

Configuring your bastion host is a never-ending process. As new vulnerabilities are discovered and new features are added to software, you will need to upgrade your host to the most current version, especially if the updated version addresses a security flaw. Along the same lines, new versions of the core OS files will be released periodically. These updates, referred to as *kernel patches*, are similar in function to some Microsoft hotfixes that address core operating system functionality. Some kernel patches will be for features or options that don't apply to you, in which case you don't need the patch. Be aware that kernel patches can be applied incrementally, as they are released, or you can download and apply the latest kernel in its entirety.

Updating Software Packages

Keeping individual software packages up to date is pretty simple, and there are a couple of approaches you can take. The most straightforward method is to use the following RPM commands to *freshen* (*-F* switch) an installed package:

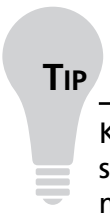
```
rpm -F apmd-3.2.2-3.2.i386.rpm
```

When you freshen a package, the newer version is installed, but only if an earlier version already exists. A good alternative is to use the *upgrade* option (by means of the *-U* switch), because this will install the new package, whether it is currently installed or not, and will remove any previous versions of the package that are present. Many experts advise simply using *–U* all the time instead of *–I* (for Install) to avoid any errors in case the package happens to already be installed on the system.

```
rpm -U apmd-3.2.2-3.2.i386.rpm
```

Updating the Kernel

There are two basic options for updating the kernel. You can download a newer kernel in its entirety and apply it, or you can download just a kernel patch and apply it. You can obtain the complete kernel or kernel patches from www.kernel.org. You should also check the Web site for your specific distribution to see if there are restrictions on which kernel will work with that distribution. In most cases you will have the choice of downloading a patch or patches or downloading the entire updated kernel. Patching the kernel generally requires a higher level of skill and a larger investment of time than installing a complete new kernel.



TIP

Kernel version numbers follow a specific pattern. There are several numbers separated by decimals, such as 2.6.17.4. In this example, the 2 represents the major version number, and the 6 represents the minor revision number. The third number represents the patch version, and the fourth is a minor revision of that patch. Essentially, the farther from the leftmost version number you get, the less significant the change. The high-level version numbers are pretty intuitive, but there is still more information hidden in a kernel version. If the minor revision number (the second number from the left) is even, that is considered a *stable* release; if it is odd, it is a *developmental* release. If the version number contains a dash, such as 2.6.17-3.4, it is a kernel that has been modified and produced by a specific distribution. The numbers to the right of the dash will follow a scheme specific to that distribution.

By entering the following command, you can determine the kernel you are currently running:

```
# uname -r
2.6.15-1.2054_FC5
```

The most current stable kernel on www.kernel.org is 2.16.17.4; however, on the Fedora site, we found the latest *Fedora Core* distribution release was 2.6.17-1.2145. Instructions on updating Fedora Core and a link to Fedora Core updates can be found at http://fedora.redhat.com/Download/updates.html. You can use Yum or RPM to update your kernel the same way you would with any other software package. One consideration is that using RPM will leave each kernel image sitting on the hard drive; over time this could account for a considerable amount of hard disk space. Yum will, by default, leave only the previous kernel in addition to the newest one and will delete all others.

Removing SUID Programs

Programs with the set user ID (SUID) bit set will be run with the permissions of the *owner*, not the user who is executing them. Obviously this could pose a big security risk. If an attacker were to locate a script with the SUID bit set and that was owned by root, and if the attacker could find a way to modify the script, he could do anything he wanted to the system, as though he were root. Although a script is the most likely target, an attacker could also gain root access if he could execute a buffer overflow against a binary program with the SUID bit set as well. The same considerations hold true for files with the set group ID (SGID) bit set. You can create a listing of all SUID and SGID files by entering the following command:

```
find / -perm -4000 –print
```

To list files with the SGID bit set, use the following command:

```
find / -perm -2000 -print
```

The list may be long, so redirecting the output to a file for review might be advisable. Unfortunately, some programs will not function properly without having the SUID or SGID bits set. Each instance will need to be researched and possibly tested in a lab environment to determine whether you can safely remove the bit. In all likelihood, either of those bits will be needed on some files, but reducing the number of files to a minimum will leave your bastion host more secure.

With all the obvious risks for using SUID and SGID programs, you might be wondering if using them is ever a good idea or simply an evil to be lived with. These bits can actually improve security in the right situation. For example, if a user needs to perform a routine maintenance action that requires root privileges, normally he needs to know the root password to temporarily gain root access and perform the task. By setting the file SUID and having it owned by root, you can give the user only permissions to execute the file but not modify it. Now he can perform the required task without ever knowing the root password.

SELinux Policy Development

SELinux, which stands for *security-enhanced Linux*, was developed in partnership with the National Security Agency (NSA). It provides a higher level of security by enforcing mandatory access control (MAC) through the kernel itself rather than by running additional processes in the user space. Because the enforcement is through the kernel, it can restrict the actions of any process, even a superuser (i.e., root) process. In fact, as far as the underlying components of SELinux are concerned, there is no concept of a root user, only security policies and security contexts. SELinux is available for many Linux distributions (see which ones at http://selinux.sourceforge.net/distros/redhat.php3) and comes included in some, such as Fedora Core. You can read the FAQ from the NSA at www.nsa.gov/selinux/info/faq.cfm for more information.

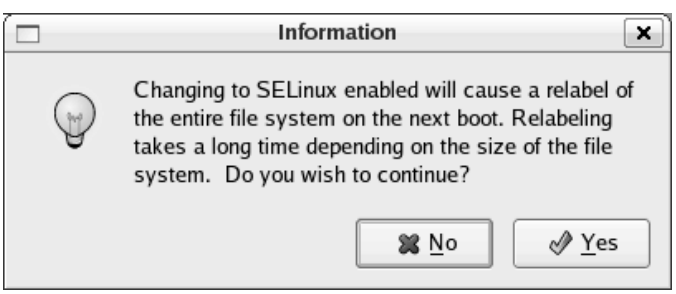
It is important to understand that SELinux is a work in progress. Currently, the setup and configuration can be rather complicated. You can enable SELinux on Fedora Core by navigating to **System | Administration | Security Level and Firewall**. On the **SELinux** tab, you must change the setting from Disabled to **Enforcing** *or* **Permissive** to enable SELinux, as shown in Figure 10.7.

Figure 10.7 Enabling SELinux



SELinux uses the *xattr* labels to generate persistent labels that describe the security context of a file or directory. These labels are not normally use; thus when you enable SELinux, you will get the warning dialog box shown in Figure 10.8. You must select **Yes** to enable SELinux, then **OK** on the **Security Level Configuration** window and reboot.

Figure 10.8 Relabel Warning Dialog Box



If you choose Permissive (which is recommended initially), the system will generate logs based on the SELinux policy but will not actually restrict any activities. This is useful for

fine-tuning the rules until you get the system into a working state. Once you are satisfied with the SELinux rules, you can enable Enforcing mode, which will actually apply your configured policy. Policies can be defined as targeted or strict. The *strict* policy applies to all processes and files on the system; the *targeted* policy is applied only to specific files. Strict, while more secure, has been found to be very difficult to configure properly. Targeted is easier to configure and is the default policy type when SELinux is first enabled. You can check the status of SELinux by typing **sestatus**.

If you navigate back to the **Security Level Configuration** window and click the **SELinux** tab, then click to expand **Modify SELinux Policy**, you will be presented with a list of options to toggle various SELinux settings. These options are only a limited set of pre-configured choices to toggle settings in the SELinux policy files. For any serious configuration you will need to edit the files manually. You can also download a third-party SELinux policy editor from the SELinux Policy Editor Project (http://seedit.sourceforge.net/index.html). This package that includes a simplified set of tools that is slightly less functional than the normal package. However, you can switch between using one or the other.

By reviewing the logs, you can see what actions the policy *would* have denied if it were enforced. If auditing has not been enabled, the *Access Vector Cache* (AVC) messages are found in /var/log/messages. If auditing has been enabled, they will be in /var/log/audit/audit.log. Further details on configuring SELinux are beyond the scope of this book. It is recommended that you read the documentation on the Web site of your chosen distribution as well as the official SELinux site (www.nsa.gov/selinux/).

TCP/IP Stack Hardening

Some means of attacking a computer system do not rely on weaknesses in the software that is providing a service but instead attack weaknesses in the underlying communications protocol. For Internet-connected systems, this means attacking weaknesses in the various protocols that make up the TCP/IP suite. Further, some of these weaknesses are intrinsic to the way the protocols work, and they cannot be removed. In those cases, you can still fine-tune the systems to make your hose more *resistant* to those attacks.

As with all security measures, they come with a cost. For example, a *SYN attack* occurs when an attacker initiates a high number of connections by sending a SYN packet, but the attacker never completes the TCP handshake. This causes the target machine to wait, with memory and CPU cycles being used on these half-open connections. The depletion of resources can cause performance degradation for all users of the system as well as reduce resources to the point where legitimate clients cannot make connections. One method of reducing your risk from such an attack is to reduce the length of time a connection can be in that half-open state before it is dropped, freeing up the resources. The downside to this approach is that legitimate clients traversing high-latency networks could have problems establishing a connection due to the short timeouts.

The way that you tune these settings, at least on some distributions, is through the *sysctl* utility. This utility allows you to configure various kernel parameters at run time. You can add settings to a script to be run at system startup, if desired. If you enter **sysctl –a | grep net.ipv4** you will see the list of available TCP/IP variables. In the previous example, to reduce the timeout of half-open connections, you would use the following command:

```
sysctl -w net.ipv4.tcp_synack_retries=3
```

The default value is 5, which means the half-open connection will not be dropped until three minutes have passed. Setting it to a value of 3 will result in only a 45-second timeout. Other values are possible, with a 1 equaling about nine seconds before timeout. As you can see, these changes will have a significant impact on the functionality of the host system, so they should be used with care. The following is a list of variables you might want to research and consider altering:

- **net.ipv4.tcp_synack_retries** Number of retransmissions in an attempt to complete the three-way handshake; a lower number is more resistant to attack.
- **net.ipv4.tcp_max_syn_backlog** Number of half-open connections that can be in the queue at any given time; a high number is more resistant to attack, at the cost of system memory.
- **net.ipv4.icmp_echo_ignore_broadcasts=1** Tells the system not to respond to a ping to a broadcast address. On systems this value is the default anyway.
- **net.ipv4.conf.all.accept_redirects=0 & net.ipv4.conf.all.send_redirects=0** Tells the system not to listen to ICMP redirects, which could be used to alter routing tables.
- **net.ipv4.conf.all.accept_source_route=0, net.ipv4.conf.all.forwarding=0 and net.ipv4.conf.all.mc_forwarding=0** Together these disable source routing, which is rarely used for anything other than attack attempts.
- **net.ipv4.conf.all.rp_filter=1** Tells the system to drop packets when the source and destination don't make sense based on the interface they came in on.
- **net.ipv4.conf.all.tcp_syncookies=1** Enables SYN cookies, allowing the connections to be made without using the backlog queue at all. If this setting is enabled, the *net.ipv4.tcp_max_syn_backlog* setting will have no effect. It is recommended to enable this setting.

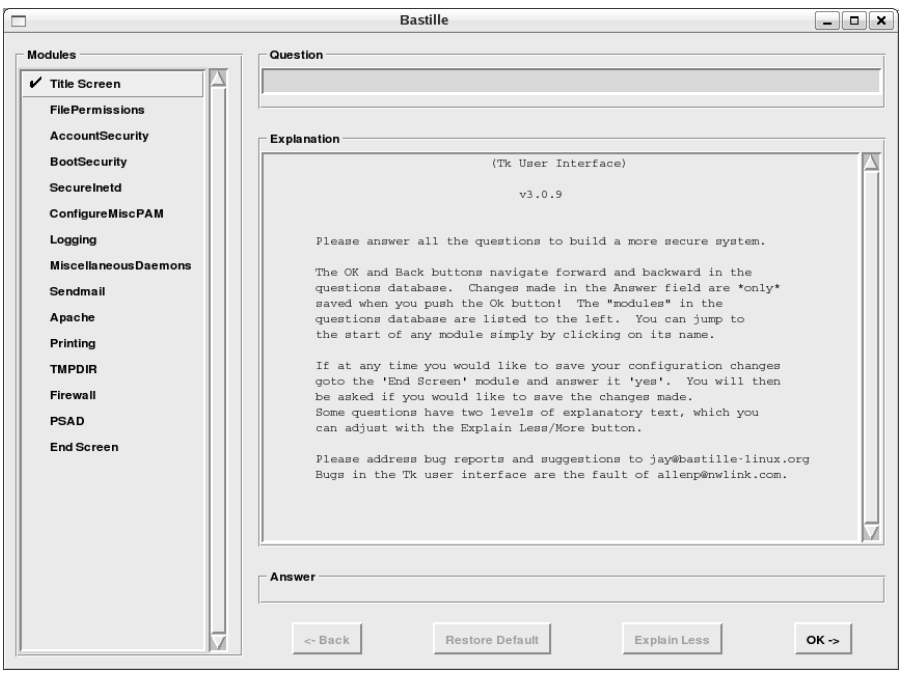
Automated Hardening Scripts

Because there are so many settings to adjust and configure, it can be easy to miss a step. As a result, many people have developed semiautomated scripts to help harden a Linux system.

The Cadillac of hardening scripts is Bastille-Linux. It has evolved from a crude basic script to a well-refined hardening system with a GUI interface. Bastille currently supports Red Hat Enterprise Linux, Fedora Core, SuSE, Debian, Gentoo, Mandrake, and HP-UX, with a Mac OS X version in development. You can read about and download Bastille from www.bastille-linux.org.

You can pass only three options to Bastille, each of which tells it to run in a different mode. If you use *bastille –report*, the system will generate a Web based report of your host's current "hardness." The *–c* option will run the actual hardening script in a text-based mode, and *–x* will run it in a graphical mode, as shown in Figure 10.9.

Figure 10.9 Bastille Graphical Configuration



All you have to do to use Bastille is start it and it will present a series of questions you must respond to with a yes or no. Based on your choices, Bastille will configure various security settings automatically. These settings include removing SUID bits from some programs, disabling insecure services (such as rshell, for example), changing user account expiration and much more. The script will usually have the most secure choice as the default selection, except in cases where the more secure selection has a high risk of impacting normal functionality. When you are finished answering the questions, you will be prompted to choose whether or not to save the resulting Bastille configuration. You will then be prompted to choose whether you want to apply the configuration or not.

Controlling Access to Resources

The majority of the steps so far have been focused on controlling *how* a user can interact with the bastion host. Most of the steps so far were made under the assumption that the user already has connectivity to the bastion host, from which programs have the SUID bit set to the security context they use and the services that are available. Following a *defense-in-depth* philosophy, the next steps are to restrict *who* can interact with the bastion host. By restricting who may communicate with the bastion host, in addition to restricting what they do once they have established communication, we come closer to having a hacker-proof system.

Address-Based Access Control

From a networking perspective, the most common way to limit access is based on the IP address of the foreign system. A traditional firewall falls into this category. In this section we discuss some of the ways you can restrict access based on the IP address of the systems in question.

Configuring TCP Wrappers

TCP Wrappers functions similarly to a firewall, except where a firewall permits or denies traffic based on data contained in the IP header of the packet, TCP Wrappers filters access to services on the host it is running on. Services that are compiled against the *libwrap.a* library can make use of TCP Wrappers. When enabled, TCP Wrappers' attempts to access a given services will be compared against the /etc/hosts.allow file and then the /etc/hosts.deny file. Rules are checked sequentially, and processing of the rules files stops when a match is found. For example, if you wanted to allow only connections from Syngress.com to SSH on your bastion host while rejecting all other attempts, you would have the following lines in your hosts.allow and hosts.deny files:

```
/etc/hosts.allow
sshd : .syngress.com

/etc/hosts.deny
sshd : ALL
```

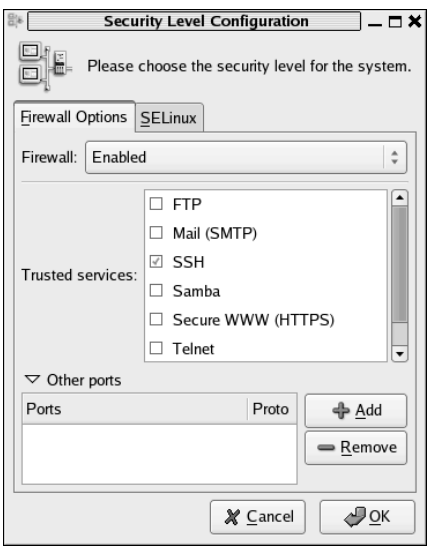
These two files accept several wildcards, such as *ALL, LOCAL, KNOWN, UNKNOWN,* and *PARANOID.* You can enable logging in the rule files as well, and configure the facility and severity of the log entry. With TCP Wrappers' limited functionality and syntax, you might wonder why you would ever use it over simply using the Linux firewall, IPTables. Because IPTables works at the packet level, if you want to deny access to a particular process, such as HTTP, you must do it based on port number. So if you use IPTables to explicitly block connection attempts to port 80, and the user starts the Web server and tells it to listen on port 8080, the connection will be allowed. With TCP Wrappers, you permit or deny access to a process. This distinction could prove invaluable if you have a service that

uses a large number of listening ports or some type of service that is spawned as needed and the port number isn't always consistent, or if the packets are tunneled in another protocol, rendering identification via port numbers impossible.

Configuring IPTables

IPTables comes with virtually every distribution of Linux. It is a very functional firewall with an impressive array of features and options. To enable the IPTables firewall, simply navigate to **System | Administration | Security Level and Firewall**. On the **Firewall Options** tab, ensure that it is **Enabled**, as shown on Figure 10.10.

Figure 10.10 Enabling an IPTables Firewall



Depending on the distribution you choose, the default rules set for IPTables may vary. In the case of Fedora Core 5, the default rule set is to permit all outbound traffic and to permit all inbound traffic that is part of an established session. As you can see in the Figure 14.10, SSH was checked. Because the default configuration blocked all inbound connection attempts, checking that box will cause a rule to be created to allow an inbound connection for SSH (TCP 22). In addition to the preconfigured ports (services), you can add your own custom port in the **Other ports** section. This interface works well for simple packet filtering, but IPTables is capable of much more functionality than mere filtering. To get more advanced, you need to be comfortable with the command line. Next we will discuss what the rules look like and their meanings.

To use an example, this is the default output from the command *iptables –L,* which is to list all chains (we will see what a chain is shortly), shown in Figure 10.11. Your default rules probably won't look exactly like these.

Figure 10.11 An IPTables Chain Listing

```
[root@localhost ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source     destination
RH-Firewall-1-INPUT  all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source     destination
RH-Firewall-1-INPUT  all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source     destination

Chain RH-Firewall-1-INPUT (2 references)
target     prot opt source     destination
ACCEPT     all  --  anywhere   anywhere
ACCEPT     icmp --  anywhere   anywhere     icmp any
ACCEPT     udp  --  anywhere   224.0.0.251  udp dpt:mdns
ACCEPT     udp  --  anywhere   anywhere     udp dpt:ipp
ACCEPT     tcp  --  anywhere   anywhere     tcp dpt:ipp
ACCEPT     all  --  anywhere   anywhere     state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere   anywhere     state NEW tcp dpt:ssh
REJECT     all  --  anywhere   anywhere     reject-with icmp-host-prohibited
[root@localhost ~]#
```

To start with, let's review some basic IPTables vocabulary. *iptables* is the command-line utility for configuring the Netfilter firewall, which is integrated with the Linux kernel. Specifically, Netfilter is actually a set of hooks inside the Linux kernel that allows the various kernel modules to interact with the network stack. In short, this is what allows the firewall to do the actual work of packet manipulation.

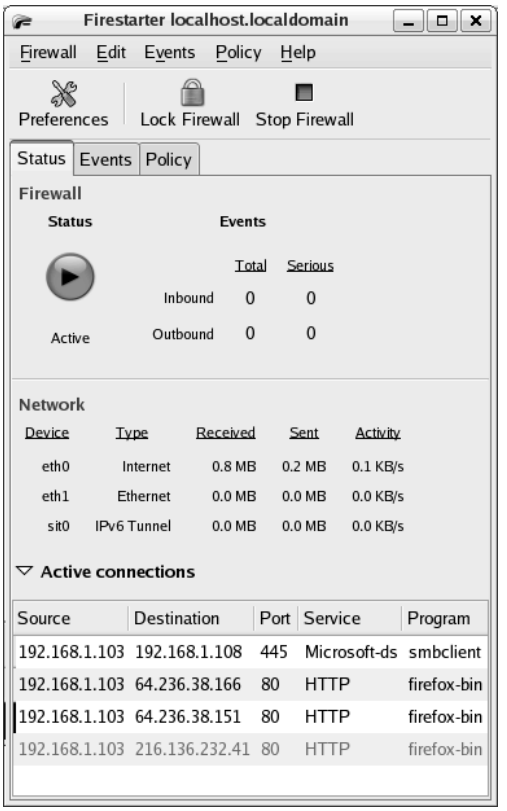
- **Tables** *NAT* (used for NAT'ing), *filter* (used for packet filtering), and *mangle* (used for other types of specialized packet modification).
- **Chains** Basically, these are access control lists that tell the firewall what to do. There are built-in chains that are automatically processed by the tables (which start out empty), and you can create your own custom chains. Built-in chains are *FORWARD, INPUT, OUTPUT, PREROUTING,* and *POSTROUTING.*
- **Targets** These dictate what action should be taken when a rule is matched. The default targets are *ACCEPT, DROP, QUEUE,* and *RETURN.*

- **Commands** These are specific *uppercase* options on the command line used to manipulate the rules themselves; for example, *–A* to append a rule to a chain, *-D* to delete a rule from a chain, and so on.
- **Options** These are used to specify options for the rule-matching criteria, such as *–s* for source address and *–p* to specify the protocol to match.

Certain tables will process certain chains by default. The one we focus on for a bastion host is the filter table. If you do not specify what table to use when adding rules (with *–t <table>*) the default is to assume you are working with the filter table. The filter table processes the *FORWARD, INPUT*, and *OUTPUT* chains by default. Referring to the chain listing, you can see that the target for the built-in chains is to jump to the custom-created chain RH-Firewall-1-INPUT. This means all packets traversing the system will go through the same chain. You don't have to set it up this way. In fact jumping out to custom chains and then back to the built-in chains can be much more efficient. As an example, let's suppose that you have a long list of rules that only need to be checked against traffic to or from the accounting subnet. Now let's place this list on its own chain called *ACCOUNTING*. You can make a single rule that matches the accounting subnet and then jumps into the *ACCOUNTING* chain containing the lengthy list of rules. In this way the majority of your traffic is only checked against the single rule matching the accounting subnet instead of the entire list of rules found in the *ACCOUNTING* chain.

The man page for IPTables contains a wealth of knowledge, and the IPTables Web site has an excellent FAQ (www.netfilter.org/documentation/index.html#documentation-faq). The large number of features and flexibility can make management over the command line sometimes cumbersome. There are several GUI tools for creating and editing the Netfilter rules. Some of the most widely used GUIs are Firestarter, Firewall Builder (*fwbuilder*), and Guarddog. These allow you to more easily create the rule set without having to know all the command-line options for IPTables. To see what some of the GUIs look like, take a look at the main window for Firestarter shown in Figure 10.12.

Figure 10.12 Main Firestarter Console



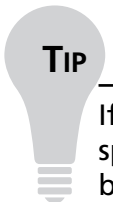
Auditing Access to Resources

Auditing serves an important role in network security. Not only does it allow you a means to know what is occurring on your systems in real time; it can also allow you to reconstruct a series of events after the fact, which can be especially important after a successful compromise has occurred. There are basically two types of auditing you can do. One is *kernel auditing*, which will log system calls, and the other is *syslog*, which can log most everything else.

Enabling the Audit Daemon

Enabling auditing is a very straightforward process. If you don't have *auditd* installed, go ahead and install it. It should be set to start automatically; if it's not, configure it to do so. The primary file for configuration of *auditd* is /etc/audit.conf. This file contains settings such as where to write the logs (defaults to /var/log/audit/audit.log), the log formatting, and what to do if the disk is full. Parameters of note are the *dispatcher, disk_full_action,* and *max_log_file_action*. Dispatcher is the program that the audit daemon will use to send all logs to *stdin*. This program will be run as root, so the program needs to be secure and used with

caution. The *disk_full_action* is exactly what it sounds like: what should the daemon do when the disk that holds the log is full? *Max_log_file_action* tells the daemon what to do when the maximum log *file size* is reached.



If *auditd* is running, the logs for SELinux are written to the same location specified in the audit.conf file. If *auditd* is not running, the logs will instead be written to the *syslog* log (which is at /var/log/messages by default). You can view the logs directly or type **dmesg** in a terminal to view them.

The other relevant file for the audit daemon is /etc/audit.rules. This file tells the audit daemon what events to audit. By default, the file contains no rules. If you wanted to audit all calls for *mount*, for example, you would place the following line in your audit.rules file:

```
-a tools,always -S mount
-w /etc/fstab
```

This basically means to add a rule to the end of the *tools* list (*-a*), and *always* generate an audit event. The *–S* means the *syscall* to watch is *mount*. *–w* tells it to *watch* the /etc/fstab file for access attempts and log them. The usefulness for being able to keep such close tabs on the use of some of the more important system calls is becoming clearer now. This is a very powerful tool for spotting intrusion attempts and other activities that could indicate someone is trying to do something they shouldn't be doing. The *watch* option does *not* accept wildcards, so *–w /etc/*.conf* will *not* work.

Enabling the Syslog Daemon

The daemon that supports *syslog* is *syslogd*. With current Linux distributions, you would be hard-pressed to find one that didn't come with *syslog* already enabled, but if you do manage to, you can install it like any other package. Once it's installed, you can control its behavior through /etc/syslog.conf. This rule file is relatively simple. Each line consists of a facility, a priority, and an action. Take the following example:

```
Mail.*          /var/log/maillog
```

This would tell the daemon to log all events from the mail subsystem, such as facility, regardless of their priority, to /var/log/maillog. Pretty straightforward. Examples of valid priorities are *debug, info, notice, crit, alert, **, and *emerg,* which indicate the overall severity of the event. An asterisk (*) stands for all facilities *or* priorities, depending on where it is used. Unless a given facility isn't used or available on your bastion host, you should point the logs someplace for all of them. The action will typically point to a real file, but some other

options are available. The most notable one from a security perspective is the *remote machine* action. This allows you to send the logs to a remote machine, which is a good idea. The syntax for the remote machine action is simply *@host*. If your bastion host were to be compromised by a hacker, any logs on that host would become suspect and of less value if legal action had to be taken. By sending the *syslog* events to a *syslog* daemon on another machine, you can ensure their integrity.

Viewing and Managing the Logs

Not that you have enabled all these logs, you will hopefully be generating a lot of useful auditing information. The issue then becomes what to do with all that data. You need a way to sort through it and pick out the most interesting pieces, then do something about it where necessary, preferably all in as automated a fashion as possible. The way to do this is through several handy log tools. One of the simplest of utilities is *dmesg*. It lacks any real features; it is merely a quick way to display the system message buffer.

A good Web site covering log analysis topics is www.loganalysis.org by Tina Bird. You can probably find a utility to do just about anything you want to do with log files, from sending e-mails for certain events to shutting down processes or just color-coding the events on the console. Two popular tools are *swatch* and *logwatch*. Of the two, *swatch* is the more lightweight, being easier to set up and having fewer options. *Swatch* is intended to parse the logs *in real time* and act on what it finds inside the logs according to the configuration you specify. *Logwatch* has a slightly different role: Its focus is on analyzing and reporting on log files, but not in real time.

Configuring Swatch

Since *swatch* (short for *simple watcher*) is relatively focused in its purpose, the setup and configuration are pretty simple. Some swatch behaviors can be set from the command line, but the rules to match must be in a configuration file. An example command line to invoke swatch would be *swatch -c /etc/swatch.conf –t /var/log/syslog,* which would tell *swatch* to use the configuration file (*-c*) at /etc/swatch.conf. If you don't specify which file to watch (*-t*), *swatch* will default to /var/log/messages or /var/log/syslog, in that order. If you don't specify a configuration file, it will echo everything to the console. You can use *–f <file>* to examine a file once instead of it running continuously with the –t options.

If you had the following lines in your configuration file, they would cause any line containing *denied* or *Denied* to echo to the console as yellow text and sound the bell once. Everything else would echo to the console as normal text.

```
watchfor      /[dD]enied/
echo yellow
bell 1
```

```
watchfor /.*/
echo
```

As you can see, configuring *swatch* is not difficult. Some of the key commands for security considerations and their use follow:

- **—script-dir=<*path to directory*>** Used on the command line. When *swatch* runs, it creates a temporary watcher script, which by default is written to the user's home directory. You should redirect the watcher script to a secured directory where it cannot be edited. A hacker with access to this temporary script could control what *swatch* reports and cover his tracks.
- **exec command** Used in the config file, this will cause matches to execute another command. This could be as simple as pager software, for example, or it could run a custom script to lock down the Netfilter firewall automatically.
- **Throttle hours:minutes:seconds,[use=message|regex|<reges>]** This is especially valuable because it controls how often duplicates of a given message will be acted on. This way a brute-force password cracker being run won't overload *swatch* with a nonstop scrolling message, possibly filling up your logging partition.
- **Threshold events:seconds,[repeat=no|yes]** This is another very important one. It allows you to ignore certain matches until they surpass a given threshold; in other words, threshold 4:60 will not perform any action unless the pattern matches four times within a 60-second window. This is very useful for things such as incorrect passwords. You don't want all sorts of alarms going off because the admin mistypes a password once, but many incorrect attempts in a short time frame may be a sign of a hacker at work.

Configuring Logwatch

Logwatch is intended to be more of a reporting tool than a live monitor. It has a host of command-line options. Logwatch will do some formatting of the output for you. For example, if you entered *logwatch —service sshd —print,* you'd get the output shown in Figure 10.13.

Figure 10.13 Output from Logwatch

```
[root@localhost ~]# logwatch --service sshd --print

 ################### LogWatch 7.1 (11/12/05) ####################
        Processing Initiated: Mon Jul 10 23:19:05 2006
        Date Range Processed: yesterday
                              ( 2006-Jul-09 )
```

```
                              Period is day.
      Detail Level of Output: 0
              Type of Output: unformatted
           Logfiles for Host: localhost.localdomain
 ##################################################################

--------------------- SSHD Begin ------------------------

SSHD Killed: 1 Time(s)

SSHD Started: 1 Time(s)

Illegal users from these:
   192.168.1.108: 4 times

Users logging in through sshd:
   root:
      192.168.1.108: 7 times

---------------------- SSHD End -------------------------


###################### LogWatch End #########################
```

As you can see, Logwatch can be invaluable in helping you sort through very large logs and extract the meaningful information in an easy-to-understand format. A command-line option of note is the *—archives —range all* option. This tells Logwatch to parse not only the log file specified but *all* archived files of that log family. For example, if used with the option *—logfile messages*, Logwatch would parse /var/log/messages in addition to /var/log/messages.* variations, such as /var/log/messages.1.

Remote Administration

It might be important to have remote administration capabilities for your bastion host. A key consideration with any administrative activities is to make sure unauthorized individuals aren't gathering information from those activities that can be used to compromise the security of the bastion host. This is true even if you are logging in directly to the console. The "guest" in the server room could be "shoulder surfing" to get your password. The same is even more true for remote administration because, by its very nature, your activities are having to traverse multiple devices and quiet probably hostile networks. Under these cir-

cumstances the key consideration is maintaining the confidentiality of the data (in this case, the administrative session) and restricting access to only authorized individuals. The challenge then becomes to provide your administrators with the needed functionality to do their jobs in a secure fashion. In this section we walk through the most common ways to accomplish this goal.

SSH

What secure shell (SSH) lacks in "flash" it makes up for in pure practicality. A secure, encrypted terminal session to the remote host is invaluable to an administrator, and with Linux's command-line-oriented design, very often no other remote access will be needed. Setting up SSH is pretty easy, too. Start off by installing the SSH daemon (*sshd*). Odds are very good it's already installed, because most distributions install it by default. Some command-line options should be configured to impact the security and operation of the daemon. This is not a complete list of options, only the most important ones from a security perspective:

- **-f <*configuration_file*>** Specifies the configuration file; *sshd* will not start without specifying a configuration file.
- **-h <host_key_file>** This option specifies the file containing the host key. You must use this option if *sshd* is *not* run as root. Because *sshd* will work without being run as root, you should configure it as a nonroot account and use this option.

The next step is to create a configuration file for *sshd*, which defaults to /etc/sshd_config if you don't specify an alternate file using the *–f* option. The following is a partial list of keywords containing only the most significant security settings. All the keywords and arguments *are* case sensitive:

- **AllowGroups** This specifies the groups allowed to log in based on group name. Wildcards are supported. The default is all groups. You should create a group specific to *sshd* users and populate it appropriately.
- **AllowUsers** This specifies the users allowed to log in based on username. The default is to allow all users. You can also specify *USER@HOST* to allow a particular user to log in only from a particular machine.
- **Banner** This is a typical banner message, which is usually a good idea to set to some ominous warning message approved by your legal department. This message is sent to the client *before* they log in and is only supported in *sshd* V2.
- **ClientAliveInterval** Specifies the time the session can be idle before *sshd* sends a message to the client requesting a response. The default is 0, which means the client

will never see these requests and subsequently never be disconnected (see below). This setting will not actually disconnect the client.

- **ClientAliveCountMax** This setting specifies how many requests for a response can be sent to the client without receiving a response before the client is disconnected. The default value is 3, so if the ClientAliveInterval were set to 20, an unresponsive client would be disconnected after 60 seconds.
- **KeyRegenerationInterval** This specifies how often to regenerate the server key. The default is one hour. If resources allow, a lower setting would be more secure, depending on how paranoid you want to be.
- **LoginGraceTime** This specifies the time the daemon will wait for a client to authenticate themselves. The default is 600 seconds, which is far too long and just asking for a DoS attack. Set this to a lower number, such as 30 seconds or less.
- **PrintMotd** This will print /etc/motd *after* the user logs in. Same considerations apply to the *Banner* option described previously.

Once you have your configuration file and the *sshd* daemon started with all the appropriate options, you need to allow inbound connection on the *sshd* listening port (default is TCP22). Using a GUI-based tool such as the Security Level Configuration utility, this is simple. Navigate to **System | Administration | Security Level and Firewall**. On the **Firewall Options** tab, next to Trusted Services, place a check mark next to **SSH**. We covered this previously in Figure 14.9. Once this is done, click **OK**, and click **OK** again when the warning dialog box comes up. To make the same changes from the command line, you could enter:

```
iptables -A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT
```

This would tell the firewall to append (*-A*) this rule to the *INPUT* chain. It would match the TCP protocol (*-p*) and look for the state to be a newly initiated connection (*-m state –state NEW*). When all this is true and the destination port (*—dport*) is 22, it will *ACCEPT* the connection (*-j ACCEPT*). Remember, the *INPUT* chain is one of the built-in chains; yours might be different. In the case of Fedora Core 5, checking the box in the GUI would add the rule to the *RH-Firewall-1-INPUT* chain instead. You can also use *—source 1.2.3.4* to specify a source address, if known, but this might not be possible if your administrators will be accessing the bastion host from home. Your system should not be able to accept connections from anywhere for SSH.

Remote GUI

Even though you can get any administrative tasks done from an SSH session, maybe you're more comfortable using a GUI. The obvious question then is, How do I access a *secure* GUI session remotely? Typically it's done one of two ways: either using X Windows' built-in func-

tionality or with third-party software such as VNC that's designed specifically to share desktops. In either case, you must encrypt the sessions somehow. The free version of VNC that's available for download, VNC Free Edition, uses an initial challenge response that hides the password, but after that the session is completely unencrypted. The current pay versions do support session encryption. Another commercial alternative is NoMachine NX server. Further information on this product can be found at www.nomachine.com. Because SSH and the X Server are a part of almost every Linux distribution and can provide the desired GUI session remotely, we will look at the settings needed to configure X Windows over SSH.

Tunneling X Windows over SSH isn't as complicated as it sounds, and if you're going to be doing the administration from a Windows machine, Cygwin/X is a free software package to provide the X Windows client and SSH client that will run on Windows. First, you need to open the appropriate ports on the Linux bastion host for inbound X Windows connections over SSH. The default SSH port is TCP port 22.

The X Server also includes its own rudimentary security mechanism. You must have a list of all user names or machine names that are allowed to connect to the X Server. This list is modified using the *xhost* utility. The process is very simple: Enter **xhost +bob** to add the *user* bob *or* the *machine* bob to the list of allowed X Server connections. Entering **xhost –bob** would remove that name from the allowed list. Finally, you need to ensure that the sshd_config must have ForwardX11 set to **Yes**. This allows the X Server to work over SSH, and in most cases it should default to **On**.

Bastion Host Configurations

Now that we have discussed the various ways to harden a Linux bastion host, it's time to do something with it. You probably don't want to have the bastion host exposed to the Internet without offering up some type of service. The most common Internet-exposed servers are FTP, Web (HTTP), mail relays (SMTP), and DNS servers. On Linux you don't need to buy any expensive software packages to have a secure, robust Web server or FTP server. Apache is the most widely used Web server on the Internet, and it's available for free. Here we examine the basic steps that are needed to provide these services securely over the Internet, and we focus on the steps that are needed *in addition* to the normal hardening procedures covered previously in this chapter.

Configuring a Web Server

Follow these steps to enable your Linux bastion host to serve as a Web server:

1. Install the Apache Web server software (*httpd*) and *openssl*. You can use any of the methods discussed previously to do so.

2. Configure the firewall to allow inbound connections to port 80 and 443 for HTTP and HTTPS, respectively. You can verify the ports are allowed by entering **iptables –L** to list the chains. There should be lines similar to the following;

```
ACCEPT     tcp  --  anywhere     start NEW tcp dpt:https
ACCEPT     tcp  --  anywhere     start NEW tcp dpt:http
```

3. If you are hosting a domain, you will want to add the domain and your server IP address to your /etc/hosts file.
4. Apache uses a configuration file to determine how it handles the Web site. You should read the documentation available at http://httpd.apache.org/docs/2.0/ for more information. The file is located at /etc/httpd/conf/http.conf by default. Edit this file if needed.
5. Assuming you want *httpd* to start automatically, go to **System | Administration | Services** and place a check next to **httpd** for the appropriate run levels.
6. Configure the appropriate monitoring for the log file (like *swatch*) if desired. The default location for the logfile is /etc/httpd/logs/.
7. At this point you should be able to connect to the Web server and see the Apache Test Page. The next step is to configure HTTPS. You can add your own content to /var/www/html and set the permissions to something appropriate for Web access, typically *read* only, or if scripts are used, *read* and *execute*; for example, **chown –R 755 /var/www/html**.
8. Install **mod_ssl**, which is the SSL/TLS module for Apache, and **openssl**.
9. Place your SSL certificate (obtained from a third-party CA) in a directory of your choice, and then edit the **/etc/httpd/conf/httpd.conf** file to include the following lines, which point to the certificate files. You should select a secured non-root partition to hold your certificate files.

```
SSLCertificateFile          <path to certificate file>/server.crt
SSLCertificateKeyFile<path to certificate file>/server.key
```

10. Edit the permissions to ensure that the server.key file is readable only by root.

Configuring an FTP Server

To configure your bastion host as an FTP server, follow these steps:

1. Select and install an FTP server daemon. Two good choices are *vsftpd* and *pure-ftpd,* but there are many others to choose from. If you are using a SELinux, *pure-ftpd* includes SELinux support, which must be downloaded separately. Some users have

reported better performance from *vsftpd,* so that is the one we will use for this example; see the *vsftpd* Web site at http://vsftpd.beasts.org/ for more information.

2. After installing the daemon, ensure that the service is set to start automatically for the appropriate run levels.
3. Enable FTP inbound on the firewall, TCP port 21. At this point you should be able to connect to the FTP server from a remote host.
4. Edit the *vsftp* configuration file (must be owned by root) located at /etc/vsftpd/vsftpd.conf. What follows are some of the more security-oriented settings to be aware of. The default settings for *vsftpd* are pretty secure.

   - **anonymous_enable** The default is Yes, and if you comment out this line, the default will still be Yes. To disable anonymous, you must set this to No.
   - **ssl_enable** This should be set to Yes to support SSL-encrypted FTP transfers.
   - **dual_log_enable** Set this to **Yes**. It tells *vsftpd* to generate a *wu-ftpd* style log (/var/log/xferlog) in addition to *vsftpd's* own logging format (/var/log/vsftpd.log). The *wu-ftpd* style log will be understood by more log-parsing utilities.
   - **force_local_logins_ssl** Setting this to Enable will force all *nonanonymous* logins to use SSL.
   - **syslog_enable** Setting this to Yes will cause the logs to be written to the system log instead.
   - **tcp_wrappers** Setting this to Yes will enable *tcp_wrapper* support (*vsftpd* has to be compiled with *tcp_wrapper* support for this to work).
   - **userlist_deny** This option is used only if *userlist_enable* is set to Yes. Setting this option to No means that a user will be denied login *unless* he or she is explicitly listed in the userlist_file (/etc/vsftpd/user_list). It's also worth noting that the user will be denied *before* being asked for a password.
   - **userlist_enable** This is the reverse of *userlist_deny*. With *user_list enable*, the user_list file is checked, and *any names in the file* will be denied access before being prompted for a password.
   - **anon_root <*path*>** This option sets the root directory for anonymous users only. This can simplify your security configuration by allowing different roots for authenticated users from anonymous users.
   - **banner_file <*path*>** This is where you specify the security warning banner.

- **ftpd_banner** This setting sets a banner message from the configu file, instead of using a separate banner file. This setting will override the banner_file setting.

After setting these directives to the desired values and restarting *vsftpd*, you should be running a secure FTP server. Remember that file permissions are of particular importance when it comes to FTP servers. You must ensure that a hacker cannot edit any sensitive files, such as configuration files. Additional security can be achieved by running your FTP server from a *chroot* jail (explained in the following DNS example).

Configuring an SMTP Relay Server

Follow these steps to secure your SMTP relay bastion host:

1. Install Sendmail or another mail transfer agent (MTA) of your choosing. We use Sendmail in this example because it is the most widely used and is included by default with many distributions. If you check your configuration, you might find it's already installed and running.
2. Edit the **/etc/mail/local-host-names** file and add to it all domains for which you want to process mail. For the relay to work, either the sender or the receiver of the mail must be in this file or you will get an error.
3. Install **sendmail-cf**.
4. Edit the **/etc/mail/sendmail.mc** file and add/configure the following lines:
   - Edit the line **dnl define('SMART_HOST',`smtp.your.provider')** with the name of your upstream mail server, and remove **dnl** from the beginning of the line.
   - Comment out the line that reads **DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA')dnl**. This line tells Sendmail to only accept mail from the local machine and is the default setting.
   - Add the following line to delete all the program and version information from the SMTP header: **define(`confSMTP_LOGIN_MSG',`')**.
   - Add the following line to remove version numbers in the HELP output; **define(`HELP_FILE',`')**.
   - Add the following line to enable privacy flags: **Define(`confPRIVACY_FLAGS',`authwarnings noexpn novrfy need-mailhelo noetrn')**.
5. Generate a new Sendmail.cf based on the new Sendmail.mc by entering **make –C /etc/mail** or **m4 /etc/mail/sendmail.mc > /etc/sendmail.cf**.

6. Add any domains you want to allow mail relay for in the **/etc/mail/access**. Use the format**somedomain.com RELAY**. After configuring this file, generate a new db for Sendmail to use by entering **makemap hash /etc/mail/access < /etc/mail/access**.
7. Start and stop Sendmail for the new settings to take effect.
8. If there are any issues (or even if there are no issues), review the mail logs at /var/log/maillog.

Configuring a DNS Server

For this example we use BIND, the de facto standard DNS server. This section assumes that you have BIND working and are familiar with DNS. Due to the complexity of configuring a DNS server, we are only going to look at the security-related settings that should be used. Follow these steps in addition to the other hardening steps that were previously discussed to get your DNS bastion host up and running.

1. Edit /etc/named.conf as follows:
   - Remove the // for **query-source address * port 53;** by default, BIND will use an unprivileged port, but often only port 53 will be allowed through many firewalls.
   - Add the option **allow-transfer { 1.2.3.4; localhost; };** to the individual zone sections. This specifies that only the machine at 1.2.3.4 and the local host are allowed to perform a zone transfer. If you do not do this, hackers will use the zone information to locate target servers and help them construct a map of your internal network.
   - In the options section (which applies to all zones), remove the ability to answer queries for domains you don't own by adding the following: **allow-query { 192.168.1.0/24; localhost; };**. This says that only queries for 192.168.1.0 and the localhost are allowed.
   - In the individual zone sections, add **allow-query { any; };**. This says that anyone is allowed to query for those specific zones and is needed to provide DNS for the domains you own.
   - In the **Options** section, disable recursive queries except from internal sources. Add the following line: **allow-recursion { 192.168.196.0/24; localhost; };**.
2. Run **named** as a nonroot user in a *chroot* jail.
   - Create a **/chroot/named** directory.

- Create the user. In this example we will create the user **named**. Create a group called **named** as well. Set the shell for the user **to /bin/false** or something equally invalid, since this user should never need to log in directly. Set the home directory to **/chroot/named**.
- Create a subdirectory structure for the jail.

```
/chroot
  +-- named
       +-- dev
       +-- etc
       |    +-- other
       |         +-- slave
       +-- var
            +-- run
```

- Move named.conf to **/chroot/named/etc** and move any zone files to **/chroot/named/etc/other**.
- Give named access to **/chroot/named/etc/other/slave** (for any zones your name server acts as a slave) and **/chroot/named/var/run** to write statistical information.
- Create a device node for /dev/null and /dev/random:

```
# mknod /chroot/named/dev/null c 2 2
# mknod /chroot/named/dev/random c 2 3
# chmod 666 /chroot/named/dev/{null,random}
```

- Copy /etc/localtime to **/chroot/named/etc/**.
- Adjust the file location in **/chroot/named/named.conf** to point to the new jail.

3. Enable logging from inside the jail by editing the startup script for *syslog* by adding **-a /chroot/named/dev/log**.
4. Modify the startup script for **named** and add **–t /chroot/named –u named**.
5. Periodically you need to update the root.hints file. This can be downloaded from ftp://ftp.internic.org/domain.named.root. Reload **named** after updating the hints file.

Bastion Host Maintenance and Support

At this point you should be comfortable with the tools required to securely administer your Linux bastion host. Ongoing maintenance must be performed on a schedule. Changes should be handled in a systematic fashion. You should define maintenance windows and a change control policy whereby patches and upgrades are approved, implemented, and reviewed. Joining one of the many vulnerability mailing lists will help you stay informed about new vulnerabilities as quickly as possible. You can bet the serious hackers are all members of those mailing lists, so you probably should be too.

There should be a policy in place for log collection and review. Logs should be recorded, preferably on a read-only media for archival purposes. Logs should be regularly reviewed and suspicious activity investigated. This last point is key; all too often logs are generated but in many cases no one looks at them. Use of automated tools like *swatch* and Logwatch can help minimize the human labor involved in log analysis, but there will still be events that need to be looked at more closely. Remember that security is an ongoing process that requires continued education and effort to stay as hacker-proof as possible.

Linux Bastion Host Checklist

Use the following checklist when hardening your Linux bastion host. These steps are not specific to any particular role (such as a web server or SMTP relay). All these steps are critical; missing any single one could leave you vulnerable to hackers and result in a compromised system.

1. Research your needs for a Linux bastion host (support, media, functionality), and select a distribution accordingly.
2. Plan the partition layout, and give some forethought to providing space for the operating system, swap partition, system logs, and system data.
3. Install the OS, and remove and disable any optional software and services.
4. Apply patches and updates to the system kernel and software as needed.
5. Remove/minimize processes using the SUID or SGID bit.
6. If mandatory access control is desired, implement SELinux.
7. Harden the TCP/IP stack.
8. Configure TCP Wrappers.
9. Configure the Netfilter firewall via the GUI or IPTables tool.
10. Apply any needed encryption for sensitive data.
11. Enable and configure auditing as required.
12. Apply scheduled maintenance to keep the system secure.

Summary

As you can see, Linux has many free and powerful security tools at its disposal. With a little care and planning, a Linux bastion host can be *at least* as secure as a Windows bastion host, if not more so. Although they may take some getting used to, the package managers offer a powerful way to install and update packages. Windows users might find the interfaces a little odd, but the ability to download free software for firewalls, Web servers, and much more, all from a single interface, is something most Windows users don't easily have. Netfilter offers a powerful firewall with many advanced features built in, for free. There is no firewall with comparable functionality included with the Windows operating system. Armed with the knowledge in this chapter, you should be able to connect your Linux bastion host to the Internet and not get hacked, and if the system should be compromised, you will have logs to see it happening, or at worst, to reconstruct events so that it doesn't happen again.

Solutions Fast Track

System Installation

- ☑ Be cognizant of alternate OS media other than a traditional hard disk install.
- ☑ Research the strengths and weaknesses of a particular distribution and choose accordingly (not all are created equally).
- ☑ Prepare the hard disk (if needed) with forethought, especially with an eye for logging.

Removing Optional Components

- ☑ Think minimally. If you don't need the software in question, remove it.
- ☑ For services you need, control when they run. Some can be enabled as needed and don't need to be running at startup.
- ☑ Ensure that none of the "high-risk services" are running at all costs, and provide comparable functionality via more secure alternatives if needed.

Additional Hardening Steps

- ☑ Make sure your system's time is accurate. This is important for forensics and encryption.

- ☑ Make sure the kernel and software are patched and up to date, to avoid known vulnerabilities.
- ☑ Remove or minimize the user of SUID or SGID files.
- ☑ Consider applying SELinux for mandatory access controls. Research the implementation details fully before coming to a decision. SELinux can offer a lot of protection, but the configuration and management can be burdensome.
- ☑ Harden the TCP/IP stack to make it more resistant to inherent weaknesses.
- ☑ Run Bastille Linux in Report mode to see if there are things you missed, or use it to make actual changes to the system.

Controlling Access to Resource

- ☑ Implement TCP Wrappers to protect processes based on client IP addresses.
- ☑ Implement the NetFilter firewall. It's free and very powerful.

Auditing Access to Resource

- ☑ Enable *auditd* for auditing of system calls.
- ☑ Enable *syslogd* for normal *syslog* logging.
- ☑ Remember to log access to the logs themselves. Attackers will often attempt to delete or edit the logs to hide evidence of their activities.
- ☑ Configure automated log monitoring to make the volume of data manageable.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: If I were to run Linux from a CD-ROM on my bastion host, won't the performance be worse than if I used a hard drive?

A: Maybe. Many of the live CDs have the option of running entirely in RAM. If you have adequate hardware, this can result in an extremely fast system. If the host will need to manipulate a lot of data, using the hard disk might be unavoidable; however, you could still run from CD-ROM and customize the distribution to make use of the hard disk for data storage.

Q: Is IPTables the firewall or Netfilter? And what is this IPChains I hear about?

A: Netfilter is the actual firewall component, though it is very common for people to refer to the firewall as IPTables. Strictly speaking, IPTables is only the command-line interface for editing the rules that Netfilter uses. IPChains is the previous incarnation of IPTables and should no longer be used if possible.

Q: Is it any more secure to use the Netfilter firewall than TCP Wrappers?

A: They don't really do the same thing. The Netfilter firewall makes filtering decisions based on the contents of the headers of the IP packets alone. TCP Wrappers makes decisions based on only the source IP and the process name that is listening for inbound connections. The functions of the two overlap only slightly. They complement each other nicely, however, and using both of them provides *defense in depth*.

Q: Why *wouldn't* I want to run X Windows remotely? I like using a GUI instead of the command line.

A: For a bastion host, the objective is to minimize exposure to attack. And like any other software running on your bastion host, X Windows is simply one more listening service that an attacker can target. Using SSH only would always be the more secure option, though X Windows is used all over the Internet today without being compromised. In the end you will just need to research and weigh the pros and cons for your environment.

Chapter 11

Apache Web Server Hardening

Solutions in this chapter:

- **Understanding Common Vulnerabilities With Apache Web Server**
- **Patching and Securing the OS**
- **Hardening the Apache Application**
- **Monitoring the Web Server for Secure Operation**

Understanding Common Vulnerabilities Within Apache Web Server

All software systems have four general types of vulnerability. Apache is no different and can be negatively affected by any one of the following problems:

- Poor application configuration
- Unsecured Web-based code
- Inherent Apache security flaws
- Foundational OS vulnerabilities

We'll investigate these four types in detail through the remaining sections of this Chapter.

Poor Application Configuration

Apache has many default settings that require modification for secure operation. Nearly all configuration information for Apache Web server exists within the httpd.conf file and associated Include files. Because many configuration options exist within these files, it can be easy to make configuration errors that expose the application to attack.

Unsecured Web-Based Code

The second manner in which vulnerabilities are exposed is via poorly implemented code on the Apache server. Often, Web developers are far more concerned with business functionality than the security of their code. For instance, poorly written dynamic Web pages can be easy DoS targets for attackers, should coded limitations be absent from backend database queries. Simply publishing confidential or potentially harmful information without authentication can provide enemies with ammunition for attack. For these reasons, you must review and understand not only the Apache application but the information and functionality being delivered via the system.

Inherent Apache Security Flaws

Like the IIS server, vulnerabilities can exist within the Apache application code itself. There are many means by which hackers can breech or disable an Apache system, such as:

- Denial of service
- Buffer overflow attacks
- Attacks on vulnerable scripts
- URL manipulation

Occasionally, Apache security flaws are discovered and announced by Apache or by various security groups. The Apache development team is typically quick to respond and distribute patches in response to such events. For this reason, it is critical that you remain vigilant in your attention to security newsgroups and to Apache's security advisory site at http://httpd.apache.org/security_report.html.

Foundational OS Vulnerabilities

Another source of vulnerability within an Apache Web server could occur as a result of foundational security flaws in the OS on which Apache is installed. Apache can be run on just about any OS. You should be very familiar with the specific security vulnerabilities for any OS on which you run Apache. This brings us to the next section, in which we discuss the merits of patching and securing the Microsoft OS.

Patching and Securing the OS

Code deficiencies can exist in OSs and lead to OS and application vulnerabilities. It is therefore imperative that you fully patch newly deployed systems and remain current with all released functional and security patches. At regular intervals, thoroughly review the published vulnerabilities at your OS manufacturer's Web site.

Some popular OSs and their respective security sites are listed in Table 11.1.

Table 11.1 Popular Operating Systems and Their Security Sites

Operating System	Vendor Security Information Site
Sun Solaris	http://sunsolve.sun.com/security
Microsoft	www.microsoft.com/technet/security/default.mspx
Mac OS	www.apple.com/support/security/
Debian Linux	www.debian.org/security/
RedHat Linux	www.redhat.com/security/
SuSe Linux	www.novell.com/linux/security/securitysupport.html
FreeBSD	www.freebsd.org/security/
NetBSD	www.netbsd.org/Security/
OpenBSD	www.openbsd.org/security.html

It might be a good idea to subscribe to your OS vendor's security mailing list to receive security-related updates and to monitor security newsgroups for 0-day exploits.

Patching Unix, Linux, and BSD Operating Systems

Because Apache is so often run on various Unix, Linux, and BSD distributions, we include patching steps so that you can confidently deploy your Apache Web server on a well-hardened foundational OS. In general, however, each vendor provides a full suite of tools and information designed to help you remain current of their released software updates. Become familiar with each of your vendor's OS patching methodologies and software tools. As the security administrator, you should reserve predetermined time periods for maintenance windows during episodes of low customer activity. However, the discovery of serious OS vulnerabilities could necessitate emergency downtime while patches are applied.

Configuring a Secure Operating System

Like patching, all systems used to provide services such as HTTP and HTTPS to customers should be thoroughly hardened *before* they are placed in a production environment. Hardening includes many steps such as the following:

- Setting file permissions
- Locking down accounts
- Establishing proper OS security policies
- Configuring host-based firewalls
- Disabling vulnerable services

Now that we have a solid, secure OS, let's move on to discuss how to properly install and securely configure the Apache Web server.

Hardening the Apache Application

The Apache Web server is a powerful application through which you can deliver critical business functionality to customers. With this power comes the possibility of misuse and attack. To ensure that your Apache server is running securely, we have compiled a series of steps to harden the Apache application. You might also want to read additional information or review other Apache security checklist documents before deploying your Apache server. Two excellent reference guides are the *CIS Apache Benchmark* document available at the Center for Internet Security (www.cisecurity.org) and the *NIST Apache Benchmark* document available at http://csrc.nist.gov/checklists/repository/1043.html.

You should follow three general steps when securing the Apache Web server as follows:

- Prepare the OS for Apache Web server
- Acquire, compile, and install the Apache Web server software
- Configure the httpd.conf file

Within these general tasks there are many steps, which we cover in the following sections.

Prepare the OS for Apache Web Server

After you'd patched and hardened your OS, you'll need to accomplish a couple quick tasks prior to obtaining, compiling, and installing the Apache software. A critical part of installing Apache is to provide a user account and group that will run the Web server. It is important that the user and group you select be unique and unprivileged to avoid reduce exposure to attack.



WARNING

Do not run your Apache Web server as the user Nobody. Although this is often a system administrator favorite and seemingly unprivileged account for running Apache and other services, the Nobody account has historically been used for root-like operations in some OSs and should be avoided.

Choose and configure a user and group account using the following Unix OS steps. In this example, we will use *wwwusr* and *wwwgrp* as the Apache username and group, respectively.

1. As root from the command line, type **groupadd wwwgrp** to add a group.
2. Type **useradd –d /usr/local/apache/htdocs –g wwwgrp –c "Apache Account" –m wwwusr** to add the user.

The second step creates the user account but also creates a home directory for the user in /usr/local/apache/htdocs.

After creating the user and group accounts, you'll need to lock down the wwwusr user account for use with Apache. By locking the account and providing an unusable shell, this action ensures that no one can actually log into the Web server using the Apache account:

1. As **root** from the command line, type **passwd –l wwwusr** to lock the Apache account.
2. Type **usermod –s /bin/false wwwusr** to configure an unusable shell account for the Apache account.

Now you're ready to get the Apache software and begin installation.

Acquire, Compile, and Install Apache Web Server Software

Because Apache is open-source software, you can freely download the binaries or source code and get going with your installation. Although there are many locations from which you could download the software, it is always best to obtain the Apache software directly from an approved Apache Foundation mirror listed at http://httpd.apache.org/download.cgi.

You'll need to decide whether to install the server using precompiled binaries or to compile the source code yourself. From a security and functionality perspective, it is usually better to obtain the source code and compile the software, since doing so permits fine-tuning of security features and business functionality. In this section, we'll discuss compiling the Apache server from source code.

To download the source code, point your browser at the URL listed previously, select a mirror, and select the latest Apache source code distribution. While you're on the Apache mirror site, also download the MD5 checksum, which should be available in the same directory as the source code. The checksum file will look identical to the source code tarball but will have an .md5 file extension. We'll use this checksum to verify the integrity of our Apache source code.

Verify Source Code Integrity

To verify the checksum, you'll need additional software called *md5sum* that might be part of your OS distribution. If it's not, you can download the software as part of GNU Textutils available at www.gnu.org/software/textutils/textutils.html. To verify the Apache checksum, perform the following steps. In this example, we'll use Apache version 2.2.3:

1. As **root** from the command line, change directories to where you downloaded the Apache source code tarball and checksum file.
2. Type **cat httpd-2.2.3.tar.gz.md5** to see the exact md5 checksum string. You should see something like **f72ffb176e2dc7b322be16508c09f63c httpd-2.2.3.tar.gz**.
3. From the same directory, type **md5sum httpd-2.2.3.tar.gz.md5** to obtain the checksum from the tarball. You should see the identical string shown in Step 2. If you do, the software you downloaded is authentic.

Compile the Source Code

After downloading and verifying the Apache source code, you'll need to do some research to understand what options you want to compile into your Web server. There are many modules, such as mod_access and mod_ssl, that can be added into your server to provide addi-

tional functionality and security. A full list of Apache Foundation-provided modules can be found at http://httpd.apache.org/docs/2.0/mod/. When choosing modules, be sure you select only what you need. Compiling extra, unnecessary modules will only result in a less secure, slower Web server.



WARNING

Use caution in enabling and disabling services at compile time. Before you do so, determine the dependencies of your Web server code. For instance, do you need CGI functionality on your site? Failure to understand what services you require to operate could result in loss of critical functionality. It might be prudent to test your configuration in a lab environment before disabling services on a production server.

Once you've decided which modules and configurations to use, you should accomplish one final task before building your software: Obscure the Apache version information located in the ap_release.h file located in the ${ApacheSrcDir}/include directory. To do so, vi the file and alter the following lines to change the Software Vendor (Apache Software Foundation) information:

```
#define AP_SERVER_BASEVENDOR "Apache Software Foundation"
#define AP_SERVER_BASEPRODUCT "Apache"
```

In general, you'll need to perform three steps to compile and install your Apache Web server, as follows:

1. From the **${ApacheSrcDir}** directory, run **./configure**.
2. After configuring source, run **./make** to compile the software.
3. After compiling the software, run **./make install** to install the Apache Web server.

During the first step, you'll decide what is added to the Apache server at compile time. Table 11.2 includes a list of modules you may consider adding and removing from your configuration.

Table 11.2 Modules You Can Add or Remove

Add/Remove	Module Name	Purpose
Remove	Status	Provides potentially dangerous information via server statistics Web page
Remove	Info	Provides potentially dangerous configuration information
Remove	Autoindex	Creates directory listing when index files are absent from Web directories
Remove	Include	Provides server-side include (SSI) functionality
Remove	Imap	Creates server-side index file mapping
Remove	userdir	Permits users to create personal homepages in ~user home directories
Remove	charset-lite	Enables character set translation
Remove	env	Modifies environment variables passed to CGI and SSI scripts
Remove	setenvif	Enables environment variable determination
Remove	asis	Permits documents to be sent without standard headers
Remove	cgi	Enables CGI scripts
Remove	negotiation	Permits standard HTTP1.1 content negotiation
Remove	actions	Permits CGI script execution
Remove	alias	Enables URL redirection and file system mapping
Add	mod_ssl	Provides cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
Add	mod_log_forensic	Increases granularity of logging to forensic levels
Add	mod_unique_id	Required for mod_log_forensic module

NOTE

To enable SSL for HTTPS operation in your Apache Web server, you'll need to download, compile, and install OpenSSL, which is available at www.openssl.org.

Mod_security

mod_security, a third-party Apache module available from www.modsecurity.org/, provides application firewall intrusion protection and prevention. To enable *mod_security*, you must download and compile the software into the Apache Web server. Adding *mod_security* increases the secure operation of your Apache Web server and adds functionality including, but not limited to, the following:

- HTTP protocol awareness
- Anti-evasion technique prevention such as URL encoding validation and URL decoding
- Enhanced audit logging
- Built-in *chroot* functionality
- Buffer overflow protection
- HTTPS filtering

We'll enable *mod_security* in our example because it adds so many security features to our system. Once you have downloaded the *mod_security* source from www.modsecurity.org/download/index.html, perform the following steps as root:

```
cd ${modsecuritySrcDir}/apache2
mkdir -r ${ApacheSrcDir}/modules/security
cp mod_security.c Makefile.in config.m4 \ ${ApacheSrcDir}/modules/security
cd ${ApacheSrcDir}
./buildconf
```

Now *mod_security* appears like other Apache modules. When we compile Apache, we'll enable it using the command *–enable-security*. Using the previous listed configurations, your *configure* statement would look something like the following:

```
./configure --prefix=/usr/local/apache \
--enable-so \
--enable-ssl \
--enable-security \
--enable-unique-id \
--enable-log-forensic \
--disable-info \
--disable-status \
--disable-autoindex \
--disable-imap \
--disable-include \
--disable-userdir \
```

```
--disable-auth \
--disable-charset-lite \
--disable-env \
--disable-setenvif \
--disable-asis \
--disable-cgi \
--disable-negotiation \
--disable-actions \
--disable-alias
```

The command configures the Apache software to be installed in /usr/local/apache and to be built with our module options. There are many options to consider in configuring the Apache source code for compilation. To view a list of options, issue the command *./configure –help* from the ${ApacheSrcDir} directory.

After successfully configuring the source code, proceed with Steps 2 and 3. On successful completion, you should see a message similar to the one shown in Figure 11.1.

Figure 11.1 Successful Completion Message

```
+--------------------------------------------------------+
| You now have successfully built and installed the      |
| Apache 2.2 HTTP server. To verify that Apache actually |
| works correctly you now should first check the         |
| (initially created or preserved) configuration files   |
|      /usr/local/apache/conf/httpd.conf                 |
| and then you should be able to immediately fire up     |
| Apache the first time by running:                      |
|      /usr/local/apache/bin/apachectl start             |
| Thanks for using Apache.        The Apache Group       |
|                                 http://www.apache.org/ |
+--------------------------------------------------------+
```

Now that we've successfully installed the Apache Web server software, let's proceed to the next step: configuring the httpd.conf file for secure operation.

Configure the httpd.conf File

The Apache Web server stores all its configuration data in the httpd.conf file located in the ${ApacheServerRoot} directory, which is, in our example, /usr/local/apache. The httpd.conf file includes many directives that can be categorized into the following sections:

- Server Directives
- User Directives
- Performance/Denial of Service (DoS) Directives
- Server Software Obfuscation Directives
- Access Control Directives
- Authentication Mechanisms
- Directory Functionality Directives
- Logging Directives

Not all directives play a significant role with regard to security. In the following sections, we'll discuss the directives that impact the security of your Apache server. Furthermore, because we disabled a lot of functionality at compile time, some directives that would normally be dangerous don't need to be removed, since they weren't added into the compiled Apache binaries. There may also be other configuration files, called Include files, associated with the httpd.conf file. Since we have enabled *mod_security*, there is a long list of potential configurations to make in an Include file called modsecurity.conf, which is usually located in the ${ApacheServerRoot}/conf directory.

In this section, we've included the Modsecurity.com-recommended *mod_security* configuration. For more information about configuring this file, refer to the *mod_security* documentation found at www.modsecurity.org/documentation/.

Recommended *modsecurity.conf* File

```
# Turn ModSecurity On
SecFilterEngine On

# Reject requests with status 403
SecFilterDefaultAction "deny,log,status:403"

# Some sane defaults
SecFilterScanPOST On
SecFilterCheckURLEncoding On
SecFilterCheckUnicodeEncoding Off

# Accept almost all byte values
SecFilterForceByteRange 1 255

# Server masking is optional
```

```
# SecServerSignature "OurServer"

SecUploadDir /tmp
SecUploadKeepFiles Off

# Only record the interesting stuff
SecAuditEngine RelevantOnly
SecAuditLog logs/audit_log

# You normally won't need debug logging
SecFilterDebugLevel 0
SecFilterDebugLog logs/modsec_debug_log

# Only accept request encodings we know how to handle
# we exclude GET requests from this because some (automated)
# clients supply "text/html" as Content-Type
SecFilterSelective REQUEST_METHOD "!^(GET|HEAD)$" chain
SecFilterSelective HTTP_Content-Type \
"!(^application/x-www-form-urlencoded$|^multipart/form-data;)"

# Do not accept GET or HEAD requests with bodies
SecFilterSelective REQUEST_METHOD "^(GET|HEAD)$" chain
SecFilterSelective HTTP_Content-Length "!^$"

# Require Content-Length to be provided with
# every POST request
SecFilterSelective REQUEST_METHOD "^POST$" chain
SecFilterSelective HTTP_Content-Length "^$"

# Don't accept transfer encodings we know we don't handle
SecFilterSelective HTTP_Transfer-Encoding "!^$"
```

User Directives

There are a couple directives you must configure in the httpd.conf file to ensure that the Apache Web server runs using the unprivileged user account we established earlier, among other things. Inspect your httpd.conf file to verify that the following statements appear as shown in the following. Recall that we decided to run Apache as wwwusr:wwwgrp.

```
User wwwusr
Group wwwgrp
```

Also, configure the ServerAdmin directive with a valid alias e-mail address such as the following:

```
ServerAdmin hostmaster@oursecuredomain.com
```

This will provide a point of contact for your customers, should they experience problems with your site.

Performance/Denial-of-Service (DoS) Directives

There are a number of performance-tuning directives in the Apache httpd.conf file. As a security professional, you should interpret these directives as DoS prevention statements, since they control resource allocation for users of the Apache server. The following directives control the performance of an Apache server:

- **Timeout** Configures the time Apache waits to receive GET requests, the time between TCP packets for POST or PUT requests, or the time between TCP ACK statements in responses. The Apache default is 300 seconds (5 minutes), but you might want to consider reducing this timer to 60 seconds to mitigate DoS attacks.
- **KeepAlive** Configures HTTP1.1-compliant persistency for all Web requests. By default, this is set to On and should remain as such to streamline Web communication.
- **KeepAliveTimeout** Determines the maximum time to wait before closing an inactive, persistent connection. Let's keep this value at the default of 15 seconds, since raising it can cause performance problems on busy servers and expose you to DoS attacks.
- **StartServers** Designates the number of child processes to start when Apache starts. Setting this value higher than the default of 5 can increase server performance, but use care not to set the value too high, because doing so could saturate system resources.
- **MinSpareServers** This setting, like the MaxSpareServers setting, allows for dynamic adjustment of Apache child processes. MinSpareServers instructs Apache to maintain the specified number of idle processes for new connections. This number should be relatively low except on very busy servers.
- **MaxSpareServers** Maintains Apache idle processes at the specified number. Like MinSpareServers, this value should be low, except for busy sites.
- **MaxClients** As its name implies, this setting determines the maximum number of concurrent requests to the Apache server. We'll leave this at the default value of 256.

Once you've finished editing this section of your httpd.conf file, you should see something similar to the following:

```
Timeout 60
KeepAlive On
KeepAliveTimeout 15
StartServers 5
MinSpareServers 10
MaxSpareServers 20
MaxClients 256
```

Server Software Obfuscation Directives

By default, Apache informs Web users of its version number when delivering a 404 (page not found) error. Since it is good practice to limit the information you provide to would-be hackers, we'll disable this feature. Recall that we already altered the Apache server signature and that we installed *mod_security*. Both of these actions should be enough to obfuscate our server because they both alter the default behavior. If you would like to turn off server signatures completely, you can always set the ServerSignature directive to Off and the ServerTokens to Prod. This will disable Apache signatures entirely.

Access Control Directives

The Apache Web server includes mechanisms to control access to server pages and functionality. The statement syntax is part of the <Directory> directive and is fairly straightforward; you specify a directory structure, whether default access is permitted or denied, and the parameters that enable access to the directory if access is denied by default. There are many options for fine-grained control that you should learn by reading http://httpd.apache.org/docs/2.0/mod/core.html#directory.

Regardless of the access you provide to your customers, you should secure the root file system using access control before placing your server into a production environment. In your httpd.conf file, you should create a statement in the access control directives area as follows:

```
<Directory />
    Order Deny, Allow
    deny from all
</Directory>
```

This statement will deny access to the root file system should someone intentionally or accidentally create a symlink to /.

Authentication Mechanisms

Apache also includes several ways in which you can authenticate customers using your Web server such as LDAP, SecureID, and basic .htaccess, to name a few examples. To use authentication mechanisms beyond basic .htaccess, you must compile additional functionality when you're building Apache. Like access control, authentication mechanisms are specified as part of the <Directory> directive.

The two steps to enabling basic .htaccess user authentication are:

1. Creating an htpasswd file to store user credentials.
2. Adding a <Directory> directive to the httpd.conf file to protect a directory structure.

Let's use an example to demonstrate how easy it can be to add authentication. In our example, we'll secure a directory called /secure and permit only customers Elise and Margot access to the files in that directory.

First, let's create an htpasswd file somewhere *not* in the Web server document root by issuing the following command:

```
htpasswd -c /usr/local/apache/passwdfile elise
New password: *****
Re-type new password: *****
Adding password for user elise
```

Next, we'll add Margot to the list as well. This time we don't need to use the *–c* argument, since our htpasswd file already exists:

```
htpasswd /usr/local/apache/passwdfile margot
New password: *****
Re-type new password: *****
Adding password for user margot
```

Now that we've established our customer accounts, we'll finish by adding a <Directory> directive to the httpd.conf file to protect the /secure directory as follows:

```
<Directory /usr/local/apache/htdocs/secure>
AuthType Basic
AuthName "Access for authenticated customers only"
AuthUserFile /usr/local/apache/passwdfile
Require user margot elise
</Directory>
```

Now, when anyone attempts to access the /secure directory, they'll be prompted for a username and password. Because we specifically require only Margot and Elise, only they will be permitted to use the directory structure, if they authenticate properly.

Let's move on to discuss a couple other security-related <Directory> directives.

Directory Functionality Directives

Within the <Directory> directive is a subdirective called *Options* that controls functionality for the directory structures specified in the <Directory> directive. The available options are listed in Table 11.3.

Table 11.3 Directory Options

Option	Functionality
All	Default setting; includes all options except MultiViews
ExecCGI	Permits CGI script execution through mod_cgi
FollowSymLinks	Allows Apache to follow OS file system symlinks
Includes	Permits SSI through mod_include
IncludesNOEXEC	Permits SSI but denies exec and exec cgi
Indexes	Allows autoindexing using mod_autoindex if no configured index file is present
MultiViews	Permits content negotiation using mod_negotiation
SimLinksIfOwnerMatch	Allows Apache to follow OS file system symlinks but only if the link and target file have the same owner

Many of the listed options are not relevant to our installation, since we disabled Includes and CGI during compile time. Regardless, a good default <Directory> directive disabling most options is shown here:

```
<Directory "/usr/local/apache/htdocs">
Order allow,deny
Allow from all
Options -FollowSymLinks -ExecCGI  -Includes -Indexes \
-MultiViews
AllowOverride None
</Directory>
```

At this point, your Apache server should be relatively secure. Let's discuss some Apache logging directives so that we can better monitor our server.

Logging Directives

There are many reasons to configure logging on your Apache server. Whether helping you see top page hits, hours of typical high volume traffic, or simply understanding who's using your system, logging plays an important part of any installation. More important, logging can

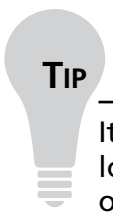
provide a near-real-time and historic forensic toolkit during or after security events. In this section, we examine some logging configuration best practices.

To ensure that your logging directives are set up correctly, we'll provide an example of the Logging options in the Apache Web server. Apache has many options with which you should familiarize yourself by reading http://httpd.apache.org/docs/2.0/mod/mod_log_config.html#logformat. This will help you understand the best output data to record in logs. Also, recall that we compiled Apache with *mod_log_forensic*, which provides enhanced granularity and logging before and after each successful page request.

An example logging configuration is shown here:

```
ErrorLog /var/log/apache/error.log
LogLevel info
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"
\"%{forensic-id}n\" %T %v" full
CustomLog /var/log/apache/access.log combined
ForensicLog /var/log/apache/forensic.log
```

This example provides a customized logging format that includes detailed output and places all the log files in the /var/log/apache directory.



TIP

It is good practice to archive your system and Apache log files to backup location. This prevents loss of critical forensic data due to accidental deletion or malicious activity.

Remove Default/Unneeded Apache Files

After you've installed and configured your Apache server, you'll need to do some quick cleanup of files that could represent a security threat. In general, you should not leave the source code you used to compile Apache on the file system. It's a good idea to tar the files up and move them to a secure server. Once you've done so, remove the source code from the Apache Web server.

You'll also want to remove some of the default directories and files installed by the Apache Web server. To do so, execute the following commands on your Web server. If you have added content into your document root directory, you'll want to avoid the first command:

```
rm -fr /usr/local/apache/htdocs/*
rm -fr /usr/local/apache/cgi-bin
```

```
rm -fr /usr/local/apache/icons
```

After removing files, let's ensure that our Apache files have proper ownership and permissions before starting our server.

Update Ownership/Permissions

As we discussed previously, the Apache Web server should be run as an unprivileged and unique account. In our example, we used the user wwwusr and the group wwwgrp to run our server. Let's make sure our permissions are properly set by running the following commands:

```
chown -R root:wwwgrp /usr/local/apache/bin
chmod -R 550 /usr/local/apache/bin
chown -R root:wwwgrp /usr/local/apache/conf
chmod -R 660 /usr/local/apache/conf
chown -R root:wwwgrp /usr/local/apache/logs
chmod -R 664 /usr/local/apache/logs
chown -R root /usr/local/apache/htdocs
chmod -R 664 /usr/local/apache/htdocs
```

Monitoring the Server for Secure Operation

Even with the best defenses and secure configurations, breeches in your systems and applications could occur. Therefore, you cannot simply set up a hardened Apache Web server and walk away thinking that everything will be just fine. Robust and comprehensive monitoring is perhaps the most important part of securely operating servers and applications on the Internet.

Throughout this book, we have discussed myriad techniques to ensure your IT security. You must leverage all these secure DMZ functions in your job. With regard to Apache, there are several things to consider that will help you identify and react to potential threats.

Your primary source of data will be through Apache and OS logs. Even with small Web sites, however, sifting through this information can be a challenge. One of the first things to consider is integrating your Apache logs with other tools to help organize and identify the potential incident "needles" in your log file "haystack." Many open-source and commercial products are available to aid you in securing your site. Once such open-source tool is called Webalizer, available at www.mrunix.net/webalizer, which features graphical representation of your Apache log file contents.

SNMP polling and graphing constitute another methodology commonly employed for secure monitoring. Often, it is extremely difficult to gauge the severity or magnitude of an event without visualization of data from logs or SNMP counters. One tool you might

consider using is a module called *mod_apache_snmp*, available at http://mod-apache-snmp.sourceforge.net/. The module can provide real-time monitoring of various metrics including, but not limited to:

- Load average
- Server uptime
- Number of errors
- Number of bytes and requests served

You might consider other commercial SNMP-based solutions, especially for enterprise-scale deployments. These tools help expedite monitoring deployment and usually include enhanced functionality to automatically alert you when important thresholds, such as Web site concurrent connections, are crossed.

Index

A

Access control, 265
ACID, 134–135, 174
Active attack, 208
Active monitoring, 136–137
Address resolution protocol, 230
Advanced Encryption Standard, 253–254
Alerting component, of Snort, 133–136
Algorithms
  Advanced Encryption Standard, 253–254
  asymmetric encryption, 255–258
  cryptographic, 255–256
  Data Encryption Standard, 252–253, 270
  definition of, 250
  description of, 250–251
  Diffie–Hellman, 256–257, 264
  digital signature, 257
  El Gamal, 257
  hashing, 258–260
  International Data Encryption Algorithm, 254
  Lucifer, 252
  Rijndael, 253–254
  Rivest, Shamir, & Adleman, 258
  strength of, 254–255
  summary of, 267
  symmetric encryption, 251–255
  Triple Data Encryption Standard, 252–253
  vulnerabilities of, 269
amap, 111–112, 118
Analyzer, 211
Anomaly detection, 123
Apache Web server
  access control directives, 396
  authentication directives, 397–398
  compiling of, 389
  default files, 399–400
  denial-of-service directives, 395–396
  description of, 35
  directory functionality directives, 398
  hardening of, 386–400
  httpd.conf file, 392–394
  installing of, 389
  logging directives, 398–399
  mod_security, 391–392
  modules, 390
  Nobody account, 387
  operating system
    preparation of, 387
    vulnerabilities of, 385–386
  performance directives, 395–396
  secure operation monitoring of, 400–401
  security flaws in, 384–385
  server software obfuscation directives, 396
  software, 388–392
  SSL enabled for, 390
  unneeded files, 399–400
  unsecured web-based code, 384
  update ownership/permissions, 400
  user directives, 394–395
  version of, 396
  vulnerabilities of, 384–385
AppArmor, 74–77, 85
Application gateways, 273
Application intrusion detection systems, 123

Application layer, of Open Systems Interconnection model, 221–223
Asymmetric cryptography, 265
Asymmetric encryption algorithms, 255–258
Atheros, 232
Attack signatures
  definition of, 123
  detection of, 123
Attacks
  brute-force, 251–252, 256
  man-in-the-middle, 262–264
  on Snort, 151–152
  SYN, 360
Audit daemon, 366–367
Auditing, 366–370
Authentication
  Apache Web server methods of, 397–398
  description of, 261, 265, 269
Authorized Use banners, 34
Automatic time synchronization, 353–355

B

Banner grabbing, 106–107
Barnyard, 174–175
BASE, 135
Basic Analysis and Security Engine, 165–172
Bastille
  configuration file for, 36
  description of, 32–33, 361
  download site for, 35
  functions of, 33–35
  graphical user interface, 37–38, 361
  implementing of, 35–41
  installation issues, 89
  logging your configurations in, 36–37
  questions, 39–40
  summary of, 84
  undoing changes, 41–42
  versions of, 35
Bastion host
  auditing access to resources, 366–370
  checklist for, 379
  configurations, 373–378
  controlling access to resources, 362–366
  description of, 342
  DNS server, 377–378
  FTP server, 374–376
  graphical user interfaces, 372–373
  hardening scripts, 361–362
  logs, 368–370
  maintenance and support of, 379
  patching of, 355
  remote administration of, 370–373
  set user ID programs, 357
  SMTP relay server, 376–377
  summary of, 380
  system installation, 342–346
  updates, 355–357
  Web server, 373–374
Benchmark tools, 79–83
Berkeley Packet Filter, 142–143
Block ciphers, 250, 256
Blocking of ports, 30–32
Bootable compact disk, 233
Broadcast domain, 226
Broadcast protocol, 212
Broadcom, 232
Brute-force attacks, 251–252, 256
Bug
  fixes for, 20, 23–25
  notification of, 21
Burned-in address, 216

C

Cable taps, 226, 231
CD-ROM installation, 280–281
Center for Internet Security, 79–83, 85
Channel bonding, 183–184
Checkpoint firewall, 278–279
Chipsets, 231
chkconfig, 348
Ciphers
  block, 250, 256
  stream, 250, 255, 269
  Vernam, 265–266
Ciphertext, 250
Cisco Security Agent, 78
clt.up file, 329
Collision domains
  definition of, 226
  hub, 226–227
  switch, 228
Commercial solutions, 7–9
CommView for WiFi, 232
Computer-based encryption, 251
Confidentiality, 261, 267, 269
Confusion operations, 253
Connectionless protocol, 218
Connection-oriented protocol, 218
Consultants, 3–4
Consulting costs, 3–4
Content scrambling system, 270
Costs
  customization, 6
  heating, ventilation, and air conditioning, 4
  maintenance, 6
  power consumption, 4
  purchase, 5
  training, 2–3
Cryptanalysis, 262
Cryptographic algorithms, 255–256
Cryptography
  access control, 265
  asymmetric, 265
  confidentiality goals, 261, 267, 269
  definition of, 250, 260
  encryption. *See* Encryption
  history of, 260–261
  integrity goals, 262–264, 269
  non-repudiation, 265
  one-time pad, 265–266, 269–270
  principles of, 262
  symmetric, 265
Cryptosystems, 262
CSMA/CD, 223–224
CTRL-ALT-DELETE rebooting, 33
Customization, 6–7

D

Daemons, 347
Data Encryption Standard, 252–253, 270
Database plug-ins, 196
Debian Linux, 385
Decryption, 250
Demarc, 135
Demilitarized zone
  description of, 7, 89
  firewalled network with, 146–147
  functions of, 145
  one-legged, 276–277
  true, 277–278
  virtual private network concentrator inside of, 327
Denial-of-service attack, 225
Denial-of-service directives, 395–396
Detection plug-ins, 195–196, 202
Diffie–Hellman algorithm, 256–257, 264
Diffusion operations, 253
Digital fingerprints, 259
Digital signature algorithm, 257

Digital signatures, 263
Disabling of services, 26–28
Discrete logarithms, 257
Distributed intrusion detection systems, 123–124
Distributions, 343
dmesg, 368
DNS, 92, 233–234, 324
DNS server, 377–378
Dsniff, 211
Dynamic host configuration protocol, 306

E

Easy Firewall Generator, 307
Edge firewall, 275
El Gamal algorithm, 257
Encrypted passwords, 259–260
Encryption
  computer-based, 251
  definition of, 250–251, 261
  opportunistic, 246
  public key, 152, 258, 263
  virtual private network, 239
Encryption algorithms
  asymmetric, 255–258
  symmetric, 251–255
Enterasys Dragon Host Sensors, 78
Enumeration
  amap, 111–112, 118
  definition of, 92
  example of, 92–93
  fingerprinting, 92, 97–98, 100–101
  httprint, 109–110, 117
  IKE-scan, 110–111, 117
  nmap: Banner Grabbing, 106–107
  remote procedure call, 97
  service identification, 96–97
  smbclient, 115–116, 118
  Windows, 112–116
  Xprobe2, 108–109, 117
  xSMBrowser, 114–115, 118
Ethernet, 212–213
EtherPeek, 211
Ettercap, 211

F

Feistel cycles, 252
File(s)
  clt.up, 329
  httpd.conf, 392–394
  srvr.up, 329
  Sudoers, 47, 50
  syslog.conf, 54
  xinetd.conf, 26–27
File transfer protocol
  description of, 220
  disabling of, 27–28
Fingerprinting, 92, 97–98, 100–101
Finisar Tap family, 226
Firestarter, 301–307
Firewall(s)
  application gateways, 273
  architectures, 274–278
  CD-ROM installation, 280–281
  checkpoint, 278–279
  description of, 272
  edge, 275
  full install, 280
  hardware, 278–279
  implementation of, 278–325
  intrusion detection system similarity to, 122
  netfilter. *See* netfilter
  packet-filtering, 272–273
  screened subnet, 274–275
  Smoothwall, 316–325
  with Snort, 145–147, 176, 202
  software, 278–279

stateful inspection, 273
summary of, 338–339
for SUSE Linux, 72–74
traffic to and from, 289–291
USB installation, 281
Windows, 291
Firewall appliance, 278
Firewall Builder, 307–316
Flow, 189
Forensics, 198
Frag3, 189
Free security solutions
commercial solutions vs., 7–9
consulting costs of, 3
costs of using, 2–5
customization costs, 6
evaluation steps for, 10–12, 16
"freeness" of, 16
hardware costs of, 3
hidden costs of, 4–5
lack of support for, 8–9
maintenance costs, 6
management capabilities of, 9
proposal for using, 14
reporting capabilities of, 9
savings associated with, 5–6
selling of, 13–14
strengths of, 7–8
testing of, 11
training costs of, 2–3
weaknesses of, 8–9
FreeBSD, 385
FTP server, 374–376

G

Gateway intrusion detection systems, 122–123
.gnmap, 99
Grabbing, 92
Graphical user interfaces
Bastille, 37–38
Firestarter, 301–307
Lokkit, 300–301
NTP, 354
remote, 372–373
Snort, 165–170

H

Haldaemon, 351
Handshaking, 226
Hardening scripts, 361–362
Hardware costs, 3
Hardware firewalls, 278–279
Hashing, 258
Hashing algorithms, 258–260
Heating, ventilation, and air conditioning costs, 4
Heuristics, 123
Home network router, 291–294
HoneyNet project, 242
Host intrusion prevention systems, 77–79, 85
Host-based intrusion detection systems, 122
Hosts, 215
HTTP, 236–237, 273
httpd.conf file, 392–394
httprint, 109–110, 117
Hubs, 159, 226

I

IBM Internet Security Systems, 78
ICMP Redirect, 230
IKE-scan, 110–111, 117
Implementation, 7
Incident handling, 198–199
Incident.pl, 135
Inefficiency-related costs, 4–5

Internal policy violators, 197
International Data Encryption Algorithm, 254
Internet control message protocol, 126
Intrusion detection systems
  application, 123
  characteristics of, 122–123
  data generated by, 125
  definition of, 122
  design of, 122
  distributed, 123–124
  firewalls and, 122, 176
  gateway, 122–123
  hacking of, 158
  host system, 158
  host-based, 122
  known-good or known-bad policy used by, 124
  limitations of, 125–126
  mechanism of action, 123–126
  network-based, 122, 158–160
  placement of, 158–160
  signature detection, 123, 137
  sniffer as, 247
  Snort. *See* Snort
  strengths of, 124–125
  summary of, 177
Intrusion prevention system, 196
IP, 224–225
IP address watchlists, 198
IPChains, 382
IPSec, 110, 240, 327
IpTables
  configuring of, 363–365
  description of, 290, 382

J

Jabber, 206
Jitter, 206

K

KeepAlive, 395
KeepAliveTimeout, 395
Kernel, 356–357
Kernel auditing, 366
Kernel patches, 355
Key exchanges, 264
Key management, 255
Keystream, 255–256
Kismet, 232

L

Lack of support, 8–9
Linux
  automatic time synchronization, 353–355
  bastion host. *See* Bastion host
  CD-ROM installation, 280–281, 344–345, 382
  disk partitions, 343
  distribution media, 344–346
  floppy disk installation, 281–282, 345
  full install, 280, 344
  installation of, 342–346
  minimizing services, 341, 347–349
  optional software, 349–352
  patching of, 386
  removal of optional components, 346–353
  security-enhanced, 63–67, 85, 357–359
  USB drive installation, 281, 345
  version of, 343
  window manager, 352–353
Log files, 56–57
Logging
  Apache Web server, 398–399
  Snort, 133–136, 138–143
  Sudo, 53–56
Logging enhancers

definition of, 57, 84
Scanlogd, 59–61, 85
SWATCH, 57–59, 85
Syslogd-ng, 61–62, 85
Loghog, 135
Logical address, 217–218
Logwatch, 368–370
Lokkit, 300–301
Lucifer algorithm, 252

M

MacSniffer, 212
Maintenance
Bastion host, 379
costs of, 6
handling of, 19–25
Mandrake Linux, 33
Man-in-the-middle attacks, 262–264
Materials, 2–3
MaxClients, 395
MaxSpareServers, 395
Media access control address
description of, 212, 227
spoofing, 230–231
Media access control sublayer, of Open Systems Interconnection model, 216–217
Message Digest 4/Message Digest 5, 260
MinSpareServers, 395
mod_security, 391–392
modsecurity.conf file, 393–394

N

nbtscan, 118
Nessus, 119
Net Optics, 226
NetBSD, 385
Netenum, 103, 117
Netfilter
CD-ROM installation, 280–281
commands, 294–296
configuring of, 279–298
description of, 279
examples of, 287–298
Filter table, 283
full install, 280
graphical user interfaces
description of, 298
Easy Firewall Generator, 307
Firestarter, 301–307
Firewall Builder, 307–316
Lokkit, 300–301
security level configuration for, 298–299
installation media, 279–298
Mangle table, 283
Nat table, 283
operation, 282–287
options summary, 296–298
packet flow, 284
Raw table, 283
rules and chains, 288–289
tables and chains, 282
TCP Wrappers vs., 382
traffic to and from the firewall, 289–291
USB installation, 281
Netstumbler, 232
Network
monitoring of, 182–183
Snort on, 136–138
virtual private. *See* Virtual private network
wireless, 231–233
Network address translation, 5, 33, 292–293
Network analysis
definition of, 204
description of, 241
summary of, 243

uses of, 207
Network analyzers
description of, 204–206
list of, 210–212
Network General Sniffer, 210
Network interface card
description of, 127
Ethernet, 216
in promiscuous mode, 212
Network policy, 241
Network-based intrusion detection systems
description of, 122
placement of, 158–160
Snort as, 143
NFS, 35
nGenius, 12
nmap
banner grabbing, 106–107
ICMP options, 99
options for, 95–96
OS fingerprinting, 100–101
output options, 99–100
Ping Sweep, 98–99
scripting, 102
speed options, 102–103
stealth scanning, 100
summary of, 116–117
Nodes, 215
NOPASSWD tag, 52–53, 90
Novell AppArmor, 74–77, 85
Novell SUSE Linux
description of, 85
firewall configuration, 72–74
logs, 71
patching of, 385
securing of, 68–74
NTP, 235–236
NX technology, 332–337, 340

O

Oinkmaster, 135, 173–174
One-legged demilitarized zone, 276–277
One-time pad, 265–266, 269–270
One-time passwords, 240
One-way functions, 259
One-way hashes, 258–259
Open Systems Interconnection model
Application layer of, 221–223
Data Link layer of, 215–217
description of, 213–215
LLC sublayer of, 217
Media Access Control sublayer of, 216–217
Network layer of, 217–218
Physical layer of, 215
Presentation layer of, 221
schematic diagram of, 214
Session layer of, 220–221
Transport layer of, 218–220
OpenBSD, 385
OpenVPN, 240
Operating systems. *See also specific operating system*
patching of, 385–386
secure, 386
updating of, 18–19
Opportunistic encryption, 246
Orinoco, 232
OS fingerprinting
passive, 107–108, 117
scanning, 100–101
Xprobe2, 108–109, 117
Output plug-ins, 196, 202

P

Package enhancements, 20
Packet sniffer, 130–131

Packet sniffing, using Snort, 138–143
Packet-filtering firewall, 272–273
Packets, 212
Packetyzer, 211
Passive attack, 208
Passive monitoring, 136
Passive network tap, 182–183
Passive OS fingerprinting, 107–108, 117
Passwords, encrypted, 259–260
Patching
  of bastion host, 355
  of Linux system, 386
  of operating systems, 385–386
  of Unix systems, 386
Penetration testing, 119
Ping Sweep, 98–99, 103
Pirut, 351
Plaintext, 250
Plug-ins, for Snort
  database, 196
  definition of, 188
  description of, 131, 156
  detection, 195–196, 202
  dynamic, 188
  output, 196, 202
  preprocessor, 188–195
Port(s)
  adding of, 31
  blocking of, 30–32
  commonly used, 219–220
  locking down, 28–32
  numbers of, 29
  switched port analyzer, 147
Port mirroring, 228–229
Port network address translation, 293
Port scanning
  description of, 94–96
  scanrand, 104–106, 117
  unicornscan, 103–104, 117
Port spanning, 228
Portscan, 189
Power consumption, 4
Preprocessors
  description of, 131–132
  plug-ins, 188–195, 202
Presentation layer, of Open Systems Interconnection model, 221
Pretty Good Privacy, 254
Prism2, 232
Private key, 257
PRTG Traffic Grapher, 175
Public key, 255, 257
Public key cryptography, 255
Public key encryption, 152, 258, 263
Public key infrastructure, 251
Pup, 351
Purchase costs, 5

R

Razorback, 135
Read-only memory, 233
Rebooting, 33
Red Hat Enterprise Linux, 346
RedHat Linux
  benchmark tools, 80–81
  description of, 33
  errata, 18
  patching of, 385
Remote access
  description of, 325
  virtual private network, 326–337
Remote Admin Trojan, 209
Remote desktop functionality, 325
Remote logging, 35
Remote procedure call enumeration, 97
Reporting, 9
Request for Comments 1700, 28
Rijndael algorithm, 253–254
Rivest, Shamir, & Adleman algorithm, 258

Rlogin service, 28
Rootkits, 208–209
Routable protocol stacks, 217
Routing, 218
Rpm, 351
r-protocols, 33
Rulesets, 132, 184–188

S

Samba, 35
Scanlogd, 59–61, 85
Scanning
  description of, 92, 94
  Fyodor's nmap. *see* nmap
  netenum, 103, 117
  port, 94–96
  speed of, 102–103
  unicornscan, 103–104, 117
scanrand, 104–106, 117
Screened subnet firewall, 274–275
Secret-key encryption, 251
Secure checksums, 259
Secure hash algorithm, 260
Secure shell, 371–372
Secure sockets layer, 240
Security advisories, 20
Security patches, 89
Security-enhanced Linux, 63–67, 85, 357–359
Sequence identification, 226
Server
  Apache Web. *See* Apache Web server
  DNS, 377–378
  FTP, 374–376
  hardening of, 25
  SMTP relay, 376–377
Server licenses, 12
Service identification, 96–97
Services
  description of, 347–348
  disabling of, 26–28
  Rlogin, 28
  stand-alone, 31–32
  Telnet, 27–28
  xinetd.conf, 26–27
Session key, 257
Session layer, of Open Systems Interconnection model, 220–221
Set user ID programs, 357
Sguil, 175
Shared-secret encryption, 251
Signature-based intrusion detection systems
  description of, 123, 137
  Snort. *See* Snort
Signatures
  community, 126
  digital, 263
smbclient, 115–116, 118
smbdumpusers, 112, 114, 118
smbgetserverinfo, 112–113, 118
smb-nat, 118
Smoothwall, 316–325
SMTP, 30, 238–239
SMTP relay server, 376–377
SneakyMan, 135
Sniffed data, 209–210
Sniffer
  "Appropriate Use" policy, 241
  creation of, 222–223
  definition of, 204
  intruder use of, 207–209
  on intrusion detection systems, 247
  one-time passwords to protect against, 240
  packet, 130–131
  protecting against, 239–241, 246
  security breach, 246

virtual private network traffic viewed using, 247
Sniffing
description of, 207–208
legal methods of, 246
mechanisms of, 212–224
packet, 138–143
wireless networks, 231–233
SNMP polling, 400
Snoop, 211
Snort
add-ons, 134–135, 172–174
alert modes, 164
alerting/logging component of, 133–136
architecture of, 129–130
attack susceptibility of, 151–152
Basic Analysis and Security Engine, 165–172
binary format of logs, 156
community signatures with, 126
configuring of, 160–172
definition of, 126, 211
description of, 2
detection engine, 132–133
effectiveness of, 175–176
e-mail alerts, 178
false alerts with, 150
features of, 129–136
firewall with, 145–147, 202
forensics, 198
graphical user interface, 165–170
hardware requirements for, 127–128
incident handling, 198–199
inline mode, 196–197
as intrusion prevention system, 196
logging, 133–136, 138–143
network architecture and, 143–149
network interface card for, 127
network pattern matching behavior, 126
network uses of, 136–138
as network-based intrusion detection systems, 143
on switched networks, 147–149
operating system requirements, 128
options, configuring of, 160–172
packet sniffer, 130–131, 138–143
pitfalls of, 149–151
plug-ins. *See* Plug-ins
policy enforcement, 197–198
preprocessor
description of, 131–132
plug-ins, 188–195, 202
rulesets, 132, 184–188
security issues with, 151–153, 179, 197–199
sensing interface, 128
sensor, 179
software with, 128–129
in subnets, 143
summary of, 154–155
system requirements for, 127–129
upgrading of, 150–151
uses of, 126
Version 2.6, 188
vulnerabilities of, 152
Snortplot.php, 134
SnortReport, 135
Snortsam, 178
SnortSnarf, 134, 175
Software firewalls, 278–279
Spoofing
definition of, 213, 225
media access control address, 230–231
srvr.up file, 329
SSH, 240
SSL VPN, 328–331
Stand-alone services, 31–32
StartServers, 395
Stateful inspection firewall, 273
Stream4, 189

Stream ciphers, 250, 255, 269
Substitution operations, 253
Sudo
  Command, 44–45
  configuring of, 47–50
  definition of, 42
  description of, 84, 89–90
  download site for, 42
  features of, 43
  installing of, 45–47
  logging, 53–56
  NOPASSWD tag, 52–53, 90
  parse errors, 47–48
  running of, 50–52
  system requirements for, 44
Sudoers file, 47, 50
Sun Remote Procedure Call service, 32
Sun Solaris, 385
SUSE Linux
  description of, 85
  firewall configuration, 72–74
  logs, 71
  patching of, 385
  securing of, 68–74
Swap space, 343
SWATCH, 57–59, 85, 368–369
Swatch, 134
Switch
  defeating of, 229–231
  definition of, 159, 227
Switch collision domains, 228
Switch flooding, 229–230
Switch ports, 231
Switched networks
  schematic diagram of, 148
  Snort on, 147–149
Switched port analyzer, 147, 160, 229
Symmetric cryptography, 265
Symmetric encryption algorithms, 251–255
SYN attack, 360
SYN scan, 94
Syslog daemon, 367–368
Syslog.conf file, 54
Syslogd-ng, 61–62, 85

T

TCP, 225–226
TCP Wrappers, 33, 152, 362–363, 382
TCPDump, 142, 211
TCP/IP
  description of, 25
  ports used by, 28–30
  stack hardening, 359–361
Technical support
  assessment of, 11
  lack of, 8–9
tee, 116, 118
Telnet
  description of, 238
  disabling of, 27–28
Testing, 11
Throttle, 59
Time synchronization, 353–355
Timestamp, 59
Time-to-live, 224
Training costs, 2–3
Transport layer, of Open Systems Interconnection model, 218–220
Transport layer security, 240
Triple Data Encryption Standard, 252–253
True demilitarized zone, 277–278

U

UDP packets, 226
UDP ports, 30
Undoing of Bastille changes, 41–42
Undo.pl, 41

unicornscan, 103–104, 117
Unix systems, 386
Update service packages, 18–19
User datagram protocol, 218

V

Vernam cipher, 265–266
Virtual private network
  assessments, 110–111
  definition of, 326
  description of, 5
  encryption and authentication used by, 239
  network design considerations, 326
  OpenSSL, 328–331
  OpenVPN, 240
  protocol used with, 327
  remote access to, 326–337
  sniffer for viewing traffic inside, 247
  SSL, 328–331
  tunnel, 326
Visudo parse error, 47–48
VLAN, 182–183
VLANACL, 182
VNC, 340

W

Web server, 373–374
Window manager, 352–353
Windows
  enumeration, 112–116
  firewalls, 291
  patching of, 385
Windows 2000 and 2003 Server Network Monitor, 211
WinDump, 210
Wireless networks, sniffing of, 231–233
Wireshark, 210, 232, 234, 242

X

X Windows, 331–337, 382
xinetd.conf, 26–27
.xml, 99
Xprobe2, 108–109, 117
xSMBrowser, 114–115, 118

Y

Yum, 351
Yum Extender, 351